Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de diplôme 2008

Filière Informatique de gestion

Web 3.0 – Déploiement OntoNostra

Etudiant : Francesco Nicola De Palma

Professeur : Anne Le Calvé

www.hevs.ch

1. TABLE DES MATIERES

2. PRÉSENTATION

2.1 DESCRIPTION DU TRAVAIL

Memoria-Mea est un projet de gestion de données personnelles composé par une multitude de modules distincts entre eux. Ces modules fournissent toutes sortes d'informations liées à un utilisateur (ses données personnelles, ses relations, ses fichiers multimédias, ses données géographiques, …) au moteur sémantique OntoMea. Ce mini-serveur se trouve en local, dans la machine de l'utilisateur, est intégré avec un moteur sémantique central OntoNostra qui lui se trouve à travers le web. Il relie un OntoMea aux autres OntoMea dans le monde.

Le web sémantique permet de mettre à disposition des données non plus exclusivement aux utilisateurs mais également aux ordinateurs. Ces données qui décrivent des données sont appelées métadonnées et sont structurées selon des normes établies par le W3C.

Actuellement OntoMea, développé par l'institut informatique de gestion de la HES-SO Valais, est un prototype de moteur sémantique permettant de gérer une base de connaissances et d'effectuer des déductions de nouvelles informations grâce à un moteur d'inférences intégré. Ce moteur déduit des informations grâce à des ontologies définies. Les informations ne se trouvent qu'en local ce qui implique plusieurs restrictions telles que :

- Un utilisateur ne peut pas y accéder lorsqu'il est sur une autre machine ou en voyage (intéressé de pouvoir déposer des informations nouvelles).
- Pas de réel partage de l'information entre utilisateurs sinon par envoi de fichiers.
- Impossibilité aux serveurs locaux de s'identifier entre eux. Par conséquence, pas de possibilité d'avoir une zone publique, privée ou semi-publique
- On pourrait profiter de la puissance des technologies du Web sémantique pour déduire de nouvelles informations lorsque l'on met ensemble des multitudes de données de plusieurs utilisateurs.

Le but de ce travail de diplôme est de choisir une architecture et de créer des web services et un démonstrateur qui passe d'un OntoMea local à un grand serveur local OntoNostra en réseau. Il devrait être possible d'accéder aux données qui sont sur OntoMea ainsi qu'à d'autres données réparties sur le Web. OntoMea est basé sur la librairie Jena (outils Java dédiés au web Sémantique) qui lit les ontologies et les métadonnées.

2.2 DEROULEMENT DU PROJET

Afin de mener à bien le travail à réaliser, la planification comporte cinq phases. Pour plus de détails sur l'emploi du temps, consulter l'annexe (*cf. : Feuille*) : Emploi du temps.

- **Une phase de recherche** et de compréhension des technologies. Cette partie, la plus importante en temps, a pour but de se familiariser avec le Web sémantique, avec le prototype d'OntoMea ainsi que l'architecture Web et la sécurité. Le cahier des charges et l'état d'art est rédigé durant cette phase. Durée : 237 heures.
- **Une phase d'analyse comprenant Installation et Configuration** des outils et des tests comparatifs. Les différentes solutions permettant de réaliser le travail sont implémentées et testées. Durée 100 heures.
- **Une phase de développement** durant laquelle les différentes fonctionnalités à apporter sont implémentées dans la solution initiale du moteur OntoNostra. Puis une application de démonstration intégrée avec des Web Services sur les points d'intérêts est réalisée. Durée 232 heures.
- **Une phase de tests et de débogage**. Durée 28 Heures.
- **Une phase de documentation** comprenant la rédaction de ce document ainsi que de ses annexes. Durée 123 heures.
- **Meeting/Email** pour contrôler le suivi du projet et échange d'information avec les responsables. Durée 27 heures

Résultat, le projet a duré environ **720** heures pour un équivalent de **90** jours pour une moyenne 20 heures par semaine. Voici un récapitulatif des heures :

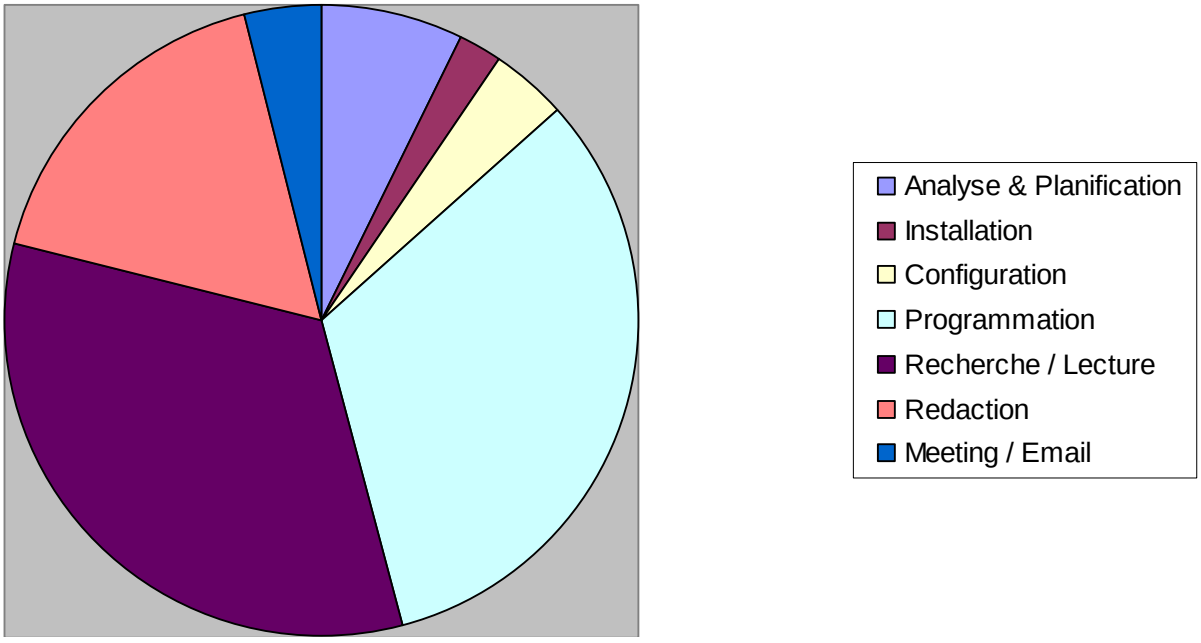


Figure 1 : Tableau des Heures

2.3 TRAVAIL A REALISER

Moteur OntoMea

Le moteur OntoMea est installé sur les postes de chaque utilisateur du Memoria-Mea en local. Il est basé sur la librairie Jena qui lit les ontologies et les données. Chaque utilisateur dispose de ses données personnelles, le serveur doit pouvoir les utiliser ainsi que d'autres données réparties sur le web.

Moteur OntoNostra

Le moteur OntoNostra devra, dans un premier temps, échanger des informations uniquement entre le serveur Central OntoNostra et les autres OntoMea (cf. Figure 2 : architecture OntoNostra). Le projet pourrait par la suite évoluer vers l'échange d'informations des OntoMea entre eux en passant toujours par OntoNostra. Le travail d'échange d'information se fera uniquement avec l'aide des Web Services qui se trouvent à la même place du serveur central. Ces Web services donneront la possibilité de travailler avec OntoNostra à tout le monde à travers le web.

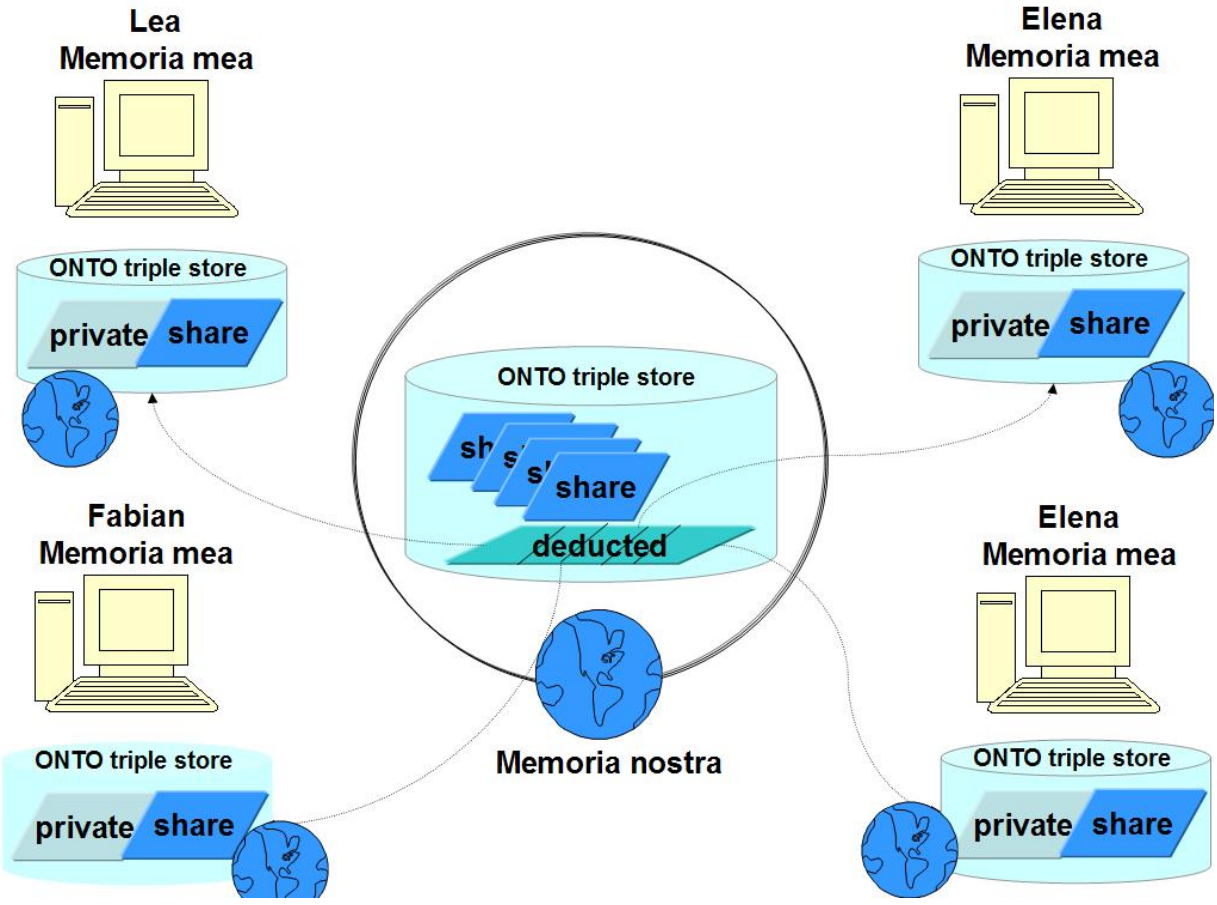


Figure 2 : architecture OntoNostra

Cet échange de données soulève plusieurs points importants à prendre en considération qui peuvent être regroupés en 5 grandes problématiques :

- Architecture
- Identification
- Authentification
- Stockage
- Sécurité

Architecture

L'architecture regroupe les éléments essentiels à prendre en considération pour l'élaboration de mon travail. Le choix de l'architecture influera sur tout le reste ou sur l'ensemble de mon travail comme le choix des outils ou des programmes utilisés dans ce travail.

Point à affronter :

- Appréhension de la problématique de l'architecture globale
- Analyse de la couche de communication entre le serveur OntoNostra et OntoMea puis implémentation de la meilleure solution (Web Service, Sémantique Web Services,…).
- Consistance et mise à jour entre les différentes localisations d'une même information.
- Utilisation du serveur central pour synchroniser différentes instances OntoMea d'une personne : ainsi des informations 'privées' peuvent transiter sur le serveur afin de mettre à jour d'autres moteurs locaux.

Identification

La base la plus importante de l'architecture Web est l'identification, l'échange d'informations étant impossible sans celle-ci. Cette identification se fera en deux grandes parties identification du serveur interlocuteur et l'identification des données envoyées ou reçues par OntoMea.

Point à affronter :

- Attribution de façon efficace des URI aux différentes ressources (données personnelles et ses relations, fichiers multimédias (image, vidéo, musique), données géographiques, …).
- Identification des OntoMea pour aussi leur attribuer des URI

Authentification

L'authentification des utilisateurs est utile aux clients qui ne possèdent pas OntoMea sur leur ordinateur. Par exemple, si les utilisateurs partent en vacances, ils voudront avoir la majorité des fonctionnalités présente sur OntoNostra à leur disposition. Afin qu'OntoNostra sache quoi faire et où placer les informations reçues il faudra que l'utilisateur s'authentifie.

Point à affronter :

- Authentification des personnes et gestion des utilisateurs (mise en place d'OpenID par exemple).
- Authentification des OntoMea par certificat

Stockage

Etant un serveur central, l'importance d'avoir un SGBD (Système de Gestion de Base de données) très performant est primordial. Les trois grandes fonctions qu'un SGBD doit avoir est l'ajout de données, mise à jour des données et la recherche des données.

Point à affronter :

- Implémentation d'un système de stockage et de mise à jour en faisant attention à la faisabilité technologique et à la performance.
- Possibilité d'ajouter des données provenant du Web.
- Gérer la mise à jour des données en tenant compte des contraintes du web sémantique
- Tests de montées en charge avec des banques de données conséquentes (GeoNames, Dbpedia,…)

Sécurité

Sans sécurité aucun utilisateur ne voudra interroger ou mettre ses données au service d'OntoNostra.

Point à affronter :

- Etude des problèmes liés à la sécurité des métadonnées et des échanges entre serveurs

2.4 STRUCTURE DU DOCUMENT

Les bases du travail et les problématique ainsi que le déroulement du projet étant été définis et structurés, il paraît essentiel à mes yeux d'expliciter brièvement la structure de ce rapport.

Dans un premier temps, le Web sémantique et ses notions sont expliqués dans un chapitre théorique intitulé ***Introduction au Web Sémantique*** avec une brève explication de quatre grands protocoles dédiés au Web 3.0 RDF, FOAF, OWL et SPARQL. Ces notions apparaissant dans tous les chapitres de ce document, il est nécessaire de le rendre plus abordable possible à ce stade. Je tiens à souligner que la plupart des termes importants sont définis très brièvement en fin de document dans la rubrique ***Glossaire***.

Ensuite, dans la partie centrale du document, je vais approfondir les problématiques listées durant le chapitre précédent « ***Présentation du Travail*** ». Je commencerai par le point le plus difficile et important ***l'architecture web*** choisie en passant par **l'*identification*** des serveurs et des ressources (données, images, personnes…), ***l'authentification*** des utilisateurs à travers le web grâce à une identifiant unique, les ***web services*** et une petite ***présentation des outils*** plus utilisés durant ce travail de diplôme.

Ultérieurement dans le classeur, le travail va aussi approfondir le ***stockage*** des données reçues et des données à envoyer ainsi que la ***sécurité*.** Pour finir la partie centrale du document, une section beacoup plus tecnique ***l'installation*** avec la ***configuration*** sera abordé plus en détail. L'installation et la configuration sera des différents outils utiles au fonctionnement du projet et à la mise en place de l'architecture Web.

En conclusion, un ***bilan*** de mon travail de diplôme réalisé avec une description des difficultés majeures et des résultats rencontrés. Les possibilités et les limites du web sémantique dans le cadre de ce projet y sont analysées. Pour finir, les différentes possibilités d'évolutions futures sont décrites et brièvement commentées. Précédé par un ***Glossaire*,** une ***Bibliographie*** avec des ***Index***, des ***Annexes*** présentant différents documents trouvés sur le web feront office. Ceux-ci se sont révélés être très utiles dans la gestion du projet de ce travail.

3. INTRODUCTION AU WEB 3. 0

« Nous allons d'un Web de documents connectés à un Web de données connectées, donc un moyen universel pour l'échange de datas ». explique **Nova Spivack**, de Radar Networks

Le Web actuel contient une quantité d'informations incroyables, cependant il demeure difficile à exploiter par les machines. Le Web est constitué par un ensemble de données, principalement textuelles, formatées dans un langage particulier (souvent HTML) permettant d'exprimer des liens, Link. Actuellement, le Web est principalement dévolu aux utilisateurs humains qui doivent analyser le contenu des pages pour déterminer sur quel lien cliquer ou non. Par exemple, si l'utilisateur cherche *Ford* c'est à lui même de différencier l'acteur de la voiture. Le moteur de recherche nous donnera toutes les pages qui contiennent *Ford* avec les liens les plus cliqués, connus mis en évidence mais, cependant, il ne pourra pas les différencier pour vous.

Pour soutenir tout cela, voilà apparaître le Web sémantique au début de ce siècle. Ce Web, également nommé Web 3.0, est une couche supplémentaire de description des données. Faite attention, le Web 3.0 n'est pas une évolution du Web 2.0, au contraire ce nom apparaît plutôt comme une moquerie envers celui-ci. Ici listé des sites typiques Web 2.0 : Facebook, Flickr, MySpace, Youtube….Alors que le Web 2.0 a surtout révolutionné la couche visible du Web et facilité les utilisateurs avec des technologies telles que les flux RSS ou AJAX, le Web 3.0 révolutionnera les couches profondes, notamment, la structure des données à travers le Web sémantique.

Qu'est-ce que le Web sémantique ? Voici une petite définition *"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation".* **Tim Berners-Lee.**

Et c'est au tour de la traduction en français : *"Le Web sémantique est une prolongation du Web actuel qui permet une définition non ambiguë de l'information pour favoriser une meilleure coopération entre les ordinateurs et les personnes ".*

Tim Berners-Lee, un des Co-inventeurs du Web, est l'un des premiers à avoir donné les principes du Web Sémantique. Il est directeur et fondateur du W3C (World Wide Web Consortium). Le **W3C** est un organisme international fondé pour promouvoir la compatibilité des projets du World Wide Web (XML, SOAP, RDF…), ce ne sont pas des lois mais des recommandations qu'ils mettent à disposition du monde entier. Site officiel: http://www.w3.org/.

Le but principal du Web sémantique est donc le développement d'un Web dont le contenu s'adresse, au moins pour une partie, aux machines, afin qu'elles puissent aider et faciliter les utilisateurs humains. Un tel Web doit doter ses ressources (documents, images, vidéo, service, pages HTML…) d'identifiants : URI et poursuit un objectif qui n'est pas seulement d'assurer l'affichage de ces ressources mais aussi de connaître son contenu en le structurant. En effet, elle ne proposera visuellement pas de modifications majeures des interfaces. Un site normal aura la même apparence qu'un site soit disant « sémantique ». Le Web sémantique est une valeur ajoutée, une extension, un travail de fond.

La puissance et l'enjeu du Web sémantique viennent de la mise en relation de la multitude de données existantes, grâce à la valorisation par la déduction de nouvelles données ainsi que de nouvelles connexions et à l'introduction au contenu existant des datas qui permettent une indexation plus poussée. Sur la base de ces inférences, de nouvelles applications aux possibilités semble-t-il infinies pourront voir le jour.

Afin d'établir une norme permettant aux différentes applications utilisant le Web sémantique de se comprendre, le W3C a publié plusieurs langages permettant de standardiser la structure des ressources. Parmi ceux-ci, on distingue deux types de langages et un autre de requête :

- Les langages permettant de décrire des données comme **RDF** (Resource Description Framework).
- Les langages permettant de structurer les données comme **OWL** (Web Ontology Language).
- Le langage permettant de faire des requêtes du type SQL comme **SPARQL**

L'objectif du projet est le partage d'informations personnelles depuis des sources locales qui se nomme OntoMea vers un système central qui est appelé OntoNostra. Le partage se fera par l'intermédiaire d'un fichier du type RDF (semblable à du XML) qui décrit la ressource mais également avec un langage de requête SPARQL propre au Web Sémantique. Dans un premier temps, il est nécessaire d'implémenter un système de stockage et de mise à jour performant des données afin d'optimiser la gestion des informations et des inférences. Ensuite, après un grand travail d'analyse des solutions existantes, un serveur central doit être mis en place avec une architecture souple et sûre. Pour l'échange entre client et serveur, des web services utiles aux OntoMea locaux pour rechercher, s'échanger les informations et l'identification des utilisateurs vont être implémenté au serveur central OntoNostra.

3.1 RDF

La force du web sémantique est de pouvoir regrouper des données provenant de différentes sources. Un RDF acronyme de *Ressource Description Framework* comme son nom l'indique est un Framework qui décrit les ressources dans le monde Web et non-Web. Ce travail va créer des métadonnées qui vont permettre de créer des liens avec des autres ressources. RDF a été conçu initialement par le W3C, basé sur une syntaxe XML et adapté à décrire une donnée et à indexer sans ambigüité les données et particulièrement les relations entre elles. Site officiel dédié à RDF : http://www.w3.org/RDF/

Le modèle des données RDF est formé d'une ressource, d'une propriété et d'une valeur. L'unité de base qui représente une information dans un RDF se compose de déclarations simples appelées **statement**. En se référant à l'image ci-dessus, Statement est composé de 3 branches {propriété, ressource, objet} que l'on nomme aussi **triplet**. *(cf. Tableau 1 : Composantes des Triplets*).

On remarque assez vite que ce triplet ressemble beaucoup au langage naturel de tous les jours comme par exemple cette phrase ***« Paris est situé en France »*** (*cf. Figure 3 : Forme d'un triplet [11]*) a la structure d'un triplet {propriété/verbe, ressource/sujet, objet/Complément d'objet}: cependant, RDF est un modèle et ne peut pas se permettre des contextes, des circonstances ou encore des conditions qu'ont en revanche les langues comme le français.

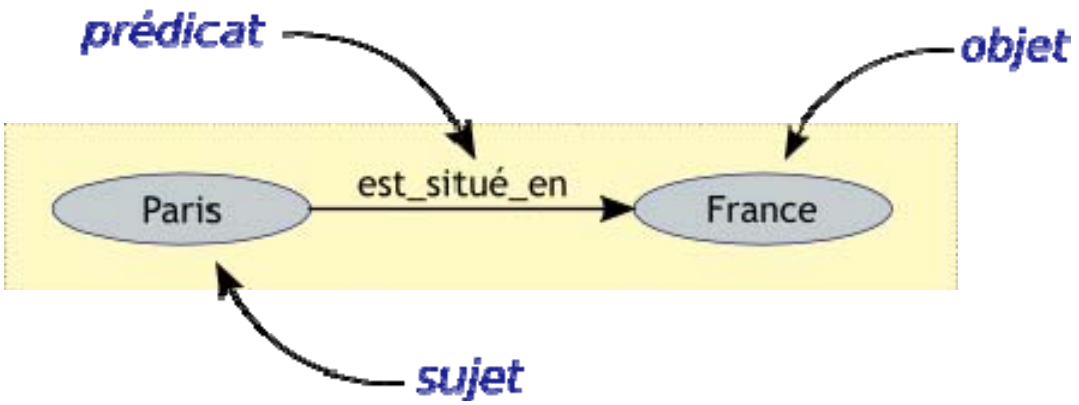


Figure 3 : Forme d'un triplet [11]

Ci-dessous présenté (*cf. Figure 4 : Triplet au format RDF/XML [11]*) un exemple de modèle utilisé par RDF, le RDF/XML qui représente la phrase énoncée juste avant : ***« Paris est situé en France ».***

```
<rdf:Description about="#paris">
 <schema:pays>France</schema:pays>
</rdf:Description>
```

Figure 4 : Triplet au format RDF/XML [11]

Sujet	Ce sont les objets décrits avec le RDF qui sont appelés ressource. Cela peut être, par exemple, une page Web entière (pour l'école dont je suis les cours : http://www.hevs.ch) ou encore une partie de page Web; (par **exemple** une partie spécifique en html ou XML http://www.hevs.ch/informatique). Cependant, il peut aussi représenter quelque chose de non accessible par le web, par exemple un livre. Ces ressources sont toujours associées par un URI.
Prédicat	Le prédicat consiste à définir un aspect spécifique, une caractéristique, un attribut ou une relation pour décrire une ressource.
Objet	L'objet indique la valeur associée au prédicat. Cela peut être, une autre ressource donc avec une autre URI, ou une valeur littérale par exemple : L'http://www.hevs.ch a 20 ans L'http://www.hevs.ch se trouve http://www.sierre.ch .

Tableau 1 : Composantes des Triplets

RDF est un modèle de données plutôt qu'un format et, en ce sens, peut donc être publié dans un grand nombre de formats différents : XML, HTML et même en texte brut. L'important est d'avoir une hiérarchie, une organisation. En effet, RDF donne la possibilité de décrire le reste des données avec des termes issus de divers vocabulaires qui répondent aux besoins spécifiques. Si on ne trouve aucun terme pour décrire les données, et bien il y aura la possibilité d'en inventer de nouveaux.

RDF n'a pas été conçu pour stocker des données. Par contre, il est mieux adapté pour offrir une grande flexibilité et une bonne interopérabilité. Ce qui signifie que les données vont être connectées avec d'autres données provenant de différentes sources, ce qui est le but du Web Sémantique. RDF sert aussi de **base** à des langages plus complexes du Web Sémantique notamment OWL, un langage dédié aux ontologies. Il va être utilisé plutôt pour l'échange, l'indexation, le formatage, les comparaisons et le traitement des données par les utilisateurs ou aussi dans notre cas par les différents serveurs locaux OntoMea. Des échanges et des chargements entre le serveur central OntoNostra et les autres OntoMea seront conclus et les fichiers RDF créés pour l'échange des informations pourront être supprimés.

Ainsi, en expliquant plus spécifiquement le fonctionnement des échanges, nous allons charger des triplets à partir de la base de connaissance. On crée un seul graph (une série de triplets) qui peut être interrogé dans son ensemble, indépendamment de la provenance des informations. De plus, on peut faire des inférences sur l'ensemble des données. Par contre, pour pouvoir gérer les mises à jour de l'information, la provenance est très importante. Par conséquent, à chaque triplet est ajoutée une quatrième information : la source du triplet, qui est nommé le **named graph**.

Par exemple (*cf. Figure 5 : Scénario Named Graph*) **:** Si la personne A et la personne B décrivent une même photo, sans même le savoir, alors les triplets créés par A et ceux créés par B sont regroupés dans un graph grâce à l'identification de la ressource. Si des données sont à double, ce n'est pas grave, ceci est géré. Par contre, le triple store se souvient de quels triplets proviennent de A et quels triplets proviennent de B. Ainsi, lorsque A modifie ses données et les renvoie au serveur, la mise à jour sera possible: on décharge le named graph de A et on charge le nouveau, sans toucher à celui de B. De cette manière, un triplet à double qui est effacé du named graph de A, ne sera alors pas perdu, parce qu'il est toujours dans le named graph de B.

Les named graphs ne devraient pas se perdre lorsque les informations transitent entre les différents OntoMea et le serveur central. Par exemple si la personne C reçoit du serveur les informations de A et de B au sujet d'une photo, les named graphs de A et B doivent être conservés pour des mises à jour futures et non pas donner aux triplets un seul named graph basé sur le serveur central (vu que les informations sont envoyées du serveur central à ce moment là).

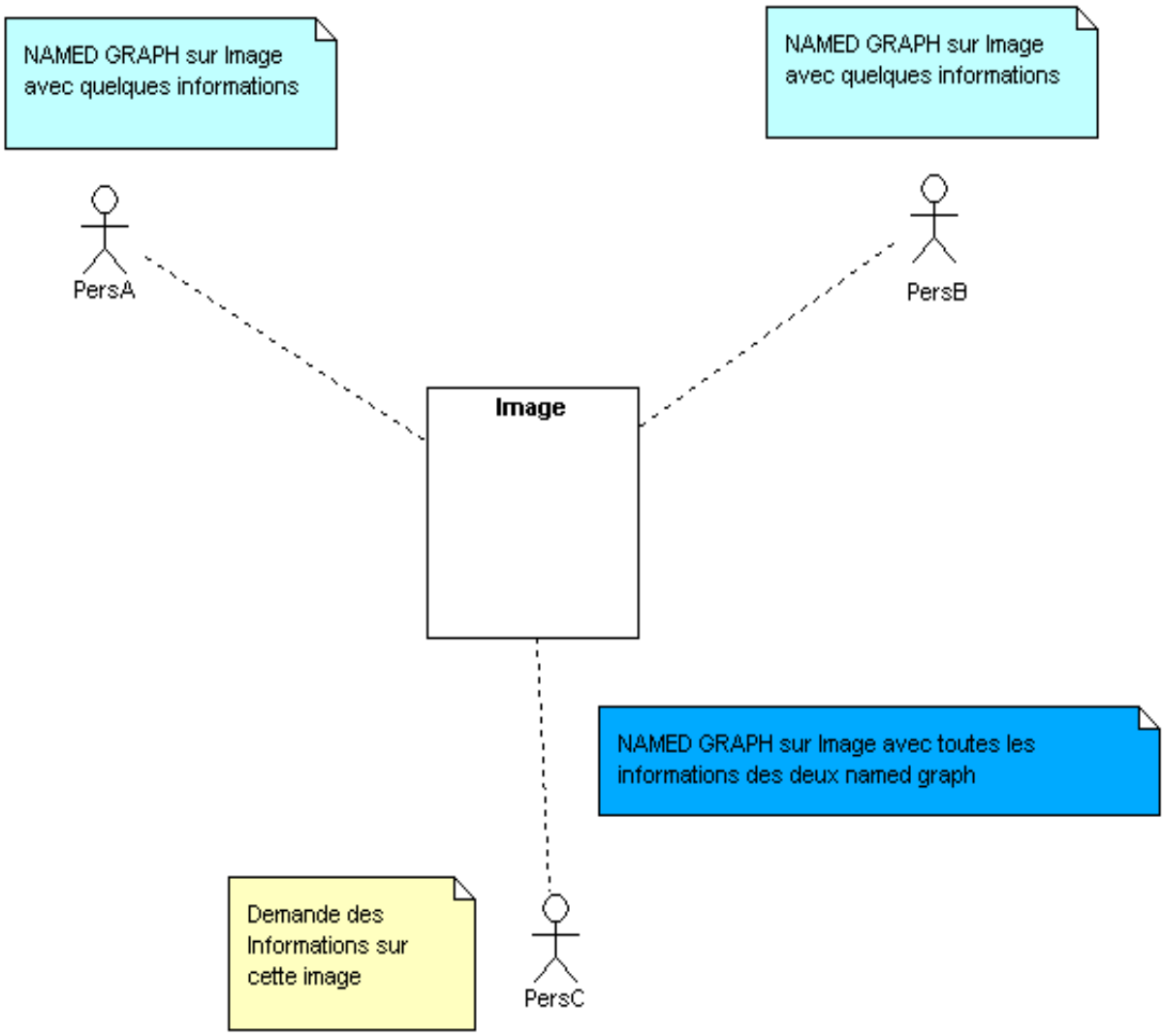
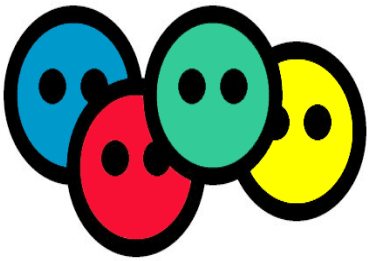


Figure 5 : Scénario Named Graph

3.2 FOAF



FOAF acronyme de ***Friend Of A Friend*** (traduit littéralement ***l'ami d'un ami***) est un projet à part du web sémantique visant à utiliser **RDF** et son fonctionnement pour créer un type de document dédié au personne. Foaf est un type de fichier accessible à travers le monde et qui décrit chaque personne et les relations qu'elles entretiennent entre elles. Il est consacré aux êtres humains et aux réseaux sociaux très en vogue actuellement depuis la venue du Web 2.0 mais aussi grâce à ce qu'elles ont fait et font. Le site officiel se trouve à cette adresse : http://www.foaf-project.org/.

Le projet utilisera ces fichiers FOAF pour échanger les données des personnes à travers les différents OntoMea. Il y a cinq grandes catégories d'informations établies par foaf :

- **Données de base** (nom, prénom, adresse ...)
- **Informations personnelles** (centres d'intérêts, connaissances…)
- **Comptes en ligne** (email, messageries instantanées…)
- **Documents et images** (textes produits par la personne, photos personnelles…)
- **Groupes et projets**

Toutes ses informations sont mises à disposition selon les trois règles suivantes :

- **Publique**, à tout le monde,
- **Semi-publique**, a un groupe restreint, par exemple, laisser son numéro de téléphone uniquement à son groupe famille
- **Privé**, à aucune personne

Cette idée sera reprise dans le fonctionnement d'OntoMea mais pas uniquement pour les humains aussi pour les fichiers multimédias (Image, Vidéo, documents). Grâce au fichier FOAF, les machines sauront regrouper de façon automatique des utilisateurs qui sans se connaître pourraient avoir les mêmes goûts musicaux, culinaire ou bien d'autres choses utiles ou futiles.

Une autre possibilité offerte par FOAF est la fusion de plusieurs documents. Supposons qu'un utilisateur crée son fichier FOAF avec OntoMea, et qu'il ne dispose pas de photos le figurant. L'un de ses amis, ayant lui aussi un fichier FOAF, indique dans celui-ci l'adresse d'une photo le montrant en compagnie de son ami. Il est alors très facile de fusionner les deux fichiers FOAF et de savoir où se trouve une image le figurant. C'est exactement le même fonctionnement du **named graph** cité dans le chapitre sur les **RDF**.

Le fichier FOAF (*cf. Figure 6 : Exemple mon fichier FOAF*), créé avec le serveur OntoMea, me décrit à travers les différentes données personnelles écrites et mises à disposition. La partie supérieure qui commence avec **<xmlns>** (XML namespace) énumère les liens vers les éléments et attributs utilisés pour créer ce fichier FOAF (RDF, OWL, SKOS, RSS, protège MEMO...). Comme exemple, l'appel à FOAF est déclaré comme ceci : <xmlns:foaf=http://xmlns.com/foaf/0.1/>

La partie inférieure, commençant par la variable **<RDF:Description>,** est la partie du fichier qui décrit la personne et ses attributs. Ce fichier créé avec OntoMea a le rôle de décrire ma personne. Par exemple, **<foaf:givenname>Francesco Nicola</foaf:givenname>** est donc la façon, du fichier FOAF, pour dire que Francesco Nicola est mon prénom, vous remarquez bien la ressemblance au fonctionnement d'XML.

```
<RDF:RDF
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:protege_dc="http://protege.stanford.edu/plugins/owl/dc/protege-
dc.owl#"
    xmlns:admin="http://webns.net/mvcb/"
    xmlns:rd="http://www.RDFdata.org/ns/"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:_8="http://gmpg.org/xfn/11#"
    xmlns:RDF="http://www.w3.org/1999/02/22-RDF-syntax-ns#"
    xmlns:swap_pim="http://www.w3.org/2000/10/swap/pim/contact#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:ical="http://www.w3.org/2002/12/cal/ical#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:food="http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#"
    xmlns:wgs84_pos="http://www.w3.org/2003/01/geo/wgs84_pos#"
    xmlns:skos="http://www.w3.org/2004/02/skos/core#"
    xmlns:imgRegion="http://www.w3.org/2004/02/image-regions#"
    xmlns:vin="http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#"
    xmlns:wot="http://xmlns.com/wot/0.1/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns="http://www.geonames.org/ontology#"
    xmlns:wn="http://xmlns.com/wordnet/1.6/"
    xmlns:rss="http://purl.org/rss/1.0/"
    xmlns:exif="http://www.w3.org/2003/12/exif/ns/"
    xmlns:vs="http://www.w3.org/2003/06/sw-vocab-status/ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:memo="http://www.memoria-mea.ch/memonto.owl#"
    xmlns:rel="http://purl.org/vocab/relationship/"
    xmlns:mindswapDM="http://www.mindswap.org/~glapizco/technical.owl#"
    xmlns:p1="http://www.domain2.com#"
    xmlns:RDFs="http://www.w3.org/2000/01/RDF-schema#" >
  <RDF:Description RDF:about="http://www.memoria-mea.ch/people/Depanico">
    <RDF:type RDF:resource="http://xmlns.com/foaf/0.1/Person"/>
    <foaf:title>Mr.</foaf:title>
    <foaf:nick>depanico</foaf:nick>
    <foaf:name>Francesco Nicola De Palma</foaf:name>
    <foaf:givenname>Francesco Nicola</foaf:givenname>
    <foaf:family_name>De Palma</foaf:family_name>
    <memo:isNowInLocation RDF:resource="http://sws.geonames.org/2658606/"/>
    <memo:isOfNationality RDF:resource="http://sws.geonames.org/3175395/"/>
    <memo:livesInLocation RDF:resource="http://sws.geonames.org/3171673/"/>
    <rel:friendOf RDF:resource="http://www.memoria-mea.ch/people/derby"/>
  </RDF:Description>
</RDF:RDF>
```

Figure 6 : Exemple mon fichier FOAF

3.3 SPARQL

SPARQL (Protocol And Query Language), comme son nom l'indique, ressemble dans l'intention à SQL. C'est un langage d'interrogation et de requêtes, mais également un protocole d'accès au RDF créé par W3C. Au contraire de SQL, il se base non pas sur une base de données relationnelle mais sur une base de triplets du type RDF. **SPARQL est «axé sur les données »** dans la mesure où il interroge uniquement l'information détenue dans les modèles, il n'existe pas de déduction dans le langage de requête elle-même. Site officiel du W3C sur SPARQL : http://www.w3.org/2001/sw/DataAccess/

SPARQL ne fait pas autre chose que de prendre la description de ce que veut l'application, sous la forme d'une requête et renvoie ces informations, sous la forme d'une série de reliures ou de graphique RDF. L'un des principaux avantages de SPARQL est de permettre de joindre des sources de données hétérogènes (ne respectant pas le même vocabulaire) très facilement.

Pour mieux approfondir, voici présenté un petit **exemple** de requête (*cf. Figure 7 : Requête SPARQL*) en SPARQL avec un SELECT très simple : **Qui est dans la liste des utilisateurs et qui a plus de 21 ans ?**

```
PREFIX foaf: http://xmlns.com/foaf/0.1/

SELECT * WHERE
{
?person foaf:name ?name.
OPTIONAL
{
?person foaf:nick ?nick
}
FILTER ( ?age > 21 )
}
ORDER BY ASC(?name)
```

Figure 7 : Requête SPARQL

Vous remarquez tout de suite la ressemblance avec SQL. Les **PREFIX** indiquent quel type de fichier sera interrogé. Dans cet exemple le fichier FOAF (un fichier RDF dédié aux personnes).

La clause **SELECT**, semblable à SQL, est utilisée pour spécifier, quelle valeur doit être retournée :

- *** :** pour toutes les variables spécifiées dans la requête,
- **?+NomVar :** pour choisir les variables:

```
SELECT ?subj ?friend ?nick WHERE
```

La clause **WHERE**, composée d'un ensemble de triplets, permet de définir les conditions dans la sélection.

SPARQL-XML dont voici le résultat reçu grâce à un fichier **XLS** (eXtensible Stylesheet Language) est un type de réponse donnée par SPARQL. XLS transforme un résultat XML/RDF dans un mode d'affichage plus facile à lire et à comprendre, pour résumer il fait le même travail d'un CSS dédié au XML.

Comme affiché dans le tableau ci dessous (cf., Tableau 2), la requête (cf. Figure 7), construite pour travailler avec **SPARQL** renvoie :

- l'URI de la **personne** (l'URI où se trouve le fichier FOAF dédié à cette personne et qu'il l'identifie),
- le **nom** de la personne
- le **nickname** choisis par toutes les personnes se trouvant sur son propre OntoMea

Person	Name	Nick
<http://www.memoria-mea.ch/people/anne>	"Anne Le CalvÃ©"	"Anna"
<http://www.memoria-mea.ch/people/fabian>	"Fabian Cretton"	"fabiany"
<http://www.memoria-mea.ch/people/Depanico>	"Francesco Nicola DePalma"	"depanico"

Tableau 2 : Résultat SPARQL

Après le tableau (cf. Tableau 2) reçu après la requête « **Qui est dans la liste des utilisateurs et qui a plus de 21 ans ? »** (cf. Figure 7 : Requête SPARQL) on va approfondir son équivalent textuel au format **RDF/XML** (cf. Figure 8 : Fichier reçu après une requête SPARQL). Le fichier XML s'organise comme tous les fichiers du même type. XML est divisé en deux grandes parties un en-tête suivi d'un corp du document). Il possède dans la partie supérieure :

- **<xmlns>** référence au namespace (des espaces de nom). Ce qui signifie que le document XML et ses variables utilisent les noms de ces namespaces identifié par leur URI.

Et par la suite, les données et les variables renvoyés par SQL qui sont construites en deux parties majeures :

- <**head**> : énumère les titres **<variable>** à renvoyer (person, name, nick). On remarque que c'est les mêmes que les en-têtes du tableau (cf Tableau 2)

- <**results**> : contient les résultats **<result>** trouvés par SPARQL avec pour chaque personne les titres énumérés dans head. Result est coupé en deux :
  - **<Binding name>** = nom de la variable (person, name, nick)
  - **<Litteral>** ou **<Uri>** = valeur de la variable

```
<?xml version="1.0"?>
<sparql
    xmlns:RDF="http://www.w3.org/1999/02/22-RDF-syntax-ns#"
    xmlns:xs="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.w3.org/2005/sparql-results#" >
  <head>
    <variable name="person"/>
    <variable name="name"/>
    <variable name="nick"/>
  </head>
  <results>
    <result>
      <binding name="person">
        <uri>http://www.memoria-mea.ch/people/anne</uri>
      </binding>
      <binding name="name">
        <literal>Anne Le Calvé</literal>
      </binding>
      <binding name="nick">
        <literal>Anna</literal>
      </binding>
    </result>
    <result>
      <binding name="person">
        <uri>http://www.memoria-mea.ch/people/fabian</uri>
      </binding>
      <binding name="name">
        <literal>Fabian Cretton</literal>
      </binding>
      <binding name="nick">
        <literal>fabiany</literal>
      </binding>
    </result>
    <result>
      <binding name="person">
        <uri>http://www.memoria-mea.ch/people/Depanico</uri>
      </binding>
      <binding name="name">
        <literal>Francesco Nicola De Palma</literal>
      </binding>
      <binding name="nick">
        <literal>depanico</literal>
      </binding>
    </result>
  </results>
</sparql>
```

Figure 8 : Fichier reçu après une requête SPARQL

3.4 ONTOLOGIES/OWL

Bien que dans mon projet je n'ai pas directement touché à la partie concernant les **Ontologies** ni aux fichiers **OWL**, il paraît tout de même important d'expliquer brièvement ces notions ainsi que la façon dont fonctionne un fichier OWL. Comme vous pouvez remarquer sur le schéma (cf. *Figure 9 : Evolution Web Sémantique [11]*), une ontologie et avec elle OWL permet de rajouter une couche supérieure de logique à toute les données et métadonnées structuré par RDF.

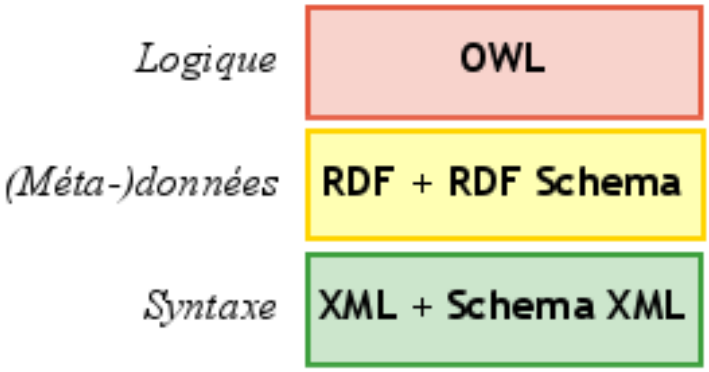


Figure 9 : Evolution Web Sémantique [11]

Définition d'Ontologie : Une ontologie permet de structurer un ensemble de concepts afin de leur donner un sens. Elle définit notamment l'ensemble de rapports possibles entre ces concepts et permet de créer des restrictions et des conditions sur ces liens en donnant en outre un vocabulaire pour les métadonnées. Grâce à la formalisation des données, l'ontologie permet une automatisation du raisonnement. Une ontologie peut être utilisée pour déduire de nouvelles informations à partir des informations disponibles. On parle alors d'inférences.

Définition d'OWL : contrairement à sa signification en anglais, OWL ne veut pas dire hibou. **Web Ontology Language** est en effet un langage basé sur RDF qui permet de définir des ontologies structurées. OWL permet d'ajouter au RDF des contraintes logiques qualifiantes comme par exemple: les propriétés de classe équivalente, de propriété équivalente, d'identité de deux ressources, de différences de deux ressources, de contraire, de symétrie, de transitivité, de cardinalité, etc., permettant de définir des rapports complexes entre des ressources.

« En conclusion, XML, RDF et OWL constituent les trois couches de base du Web Sémantique : XML est le support de sérialisation sur lequel s'appuient RDF et OWL pour définir des structures de données et les relations logiques qui les lient. Concrètement, cela signifie qu'on peut exprimer et structurer des concepts complexes à l'aide de RDF et OWL, là où jusqu'à présent on avait recours aux bases de données relationnelles. RDF et OWL sont en effet deux langages XML, conçus spécialement pour l'énonciation de faits atomiques » [1]

Site officiel : http://www.w3.org/2004/OWL/ et traduction autorisé en français http://www.yoyodesign.org/doc/w3c/owl-guide-20040210/#Introduction

4. ARCHITECTURE

Après avoir dégrossi une grande partie de théorie du Web Sémantique et de ses types de langages, c'est au tour d'abordé à ce chapitre : **l'architecture**. Le point central dans le projet, bien sûre après le monde Web 3.0, est la mise en place d'une architecture Web adapté au sémantique et à la discussion entre OntoMea et le serveur central OntoNostra. Le choix doit pouvoir mettre en œuvre les points importants que le Web Sémantique et sa théorie préconise pour le faire fonctionner au mieux. Comment choisir la meilleure architecture adaptée aux besoins du projet et de ses éléments. Voici listés les points à affronter repris du chapitre **Description du travail**:

- Appréhension de la problématique de l'architecture globale
- Analyse de la couche de communication entre le serveur OntoNostra et OntoMea puis implémentation de la meilleure solution (Web Service, Sémantique Web Services,…).
- Consistance et mise à jour entre les différentes localisations d'une même information.
- Utilisation du serveur central pour synchroniser les différentes instances OntoMea d'une personne : ainsi des informations 'privées' peuvent transiter sur le serveur afin de mettre à jour d'autres moteurs locaux.

Ce sont tout les points qui ont été affrontés et qui seront développés à travers les sections qui vont suivre. Bien sûre, il est essentiel de partir comme même d'une base sûre pouvant permettre au mieux l'élaboration de mon travail de diplôme. L'architecture du projet requiert bien une meilleure réutilisabilité de l'infrastructure existante et nécessite de favoriser l'interopérabilité entre OntoNostra et OntoMea. Pout tout cela il convient de concevoir une architecture distribuée qui utilise les Web Services sur http(s). Les services Web sont des minis applications offrant un « service » qui s'éxécutent via Interner ou Intranet, voir section 7 pour une plus ample définition avec aussi une explication plus approfondi des services offerts par OntoNostra.

Dans les deux prochaines sections, le document va tenter d'approfondir les choix faits durant le Travail de diplôme en matière d'architecture Web. La séléction a été en fait entre les deux principaux types d'architecture :

- Une architecture **client/serveur,** un serveur central et plein de petits clients qui ne peuvent travailler qu'avec lui qui ne peuvent communique entre eux. (*cf. Figure 10 : Client/serveur vs P2P– schéma de gauche)*
- Une architecture **p2p**, où toutes les machines ont les mêmes fonctionnalités (tous des serveurs ou tous des clients), il n'y a aucune machine indispensable. (*cf. Figure 10 : Client/serveur vs P2P– schéma de droite)*

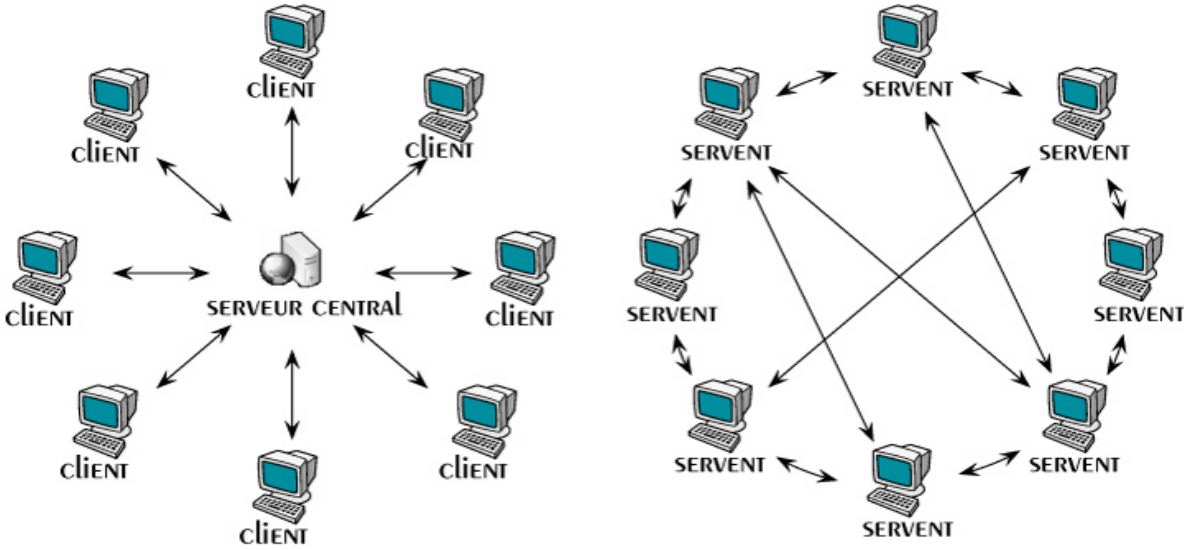


Figure 10 : Client/serveur vs P2P [12]

Pour conclure dans la dernière section de ce chapitre, le rapport présentera et développera l'idée d'un style d'architecture : **REST**. Un style d'architecture est une série de régle et de contraintes à mettre en place dans une architecture. Rest a été choisi pour sa similitude au Web Sémantique. Durant cette section, une définition plus poussés sera mis en place avec une similitude qu'il posséde avec le Web 3.0. A la fin de cette section dédiée à Rest, le document va aussi approfondir, dans une mini-section, Http avec la définition des codes qu'il utilise. L'explication d'http est très importante car Rest s'y appuie beaucoup.

4.1 SERVEUR CENTRAL

En faisant référence à la figure présentée au chapitre sur la **présentation du projet**, je suis parti de l'architecture la plus connue et plus utilisée : une solution à étoile appelé Client/serveur. (*cf. Figure 10 : Client/serveur vs P2P– schéma de gauche*).

- Les **clients** sont des applications ou des machines qui cherchent à exploiter des services comme (Web Services, envois de mails, requêtes SQL ou bien même affichage d'une page Web...) proposés par un serveur distant en particulier dans notre architecture à travers le Web
- Les **serveurs** sont des machines plus puissantes que des clients qui offrent ces services et qui identifient les clients. Les serveurs peuvent avoir différents rôles (l'accès aux utilisateurs, l'identification des clients, partage et sécurité des données).

Les clients et le serveur sont liés par des protocoles de communication, sécurisés ou non, à travers un réseau de communication. Les protocoles tels que : **http, ftp, smtp** sont des normes qui permettent à deux machines de se comprendre et de parler la même « langue » même si c'est des machines différentes et lointaine. Le réseau de communication peut être le web ou même un réseau local comme, par exemple, entre un ordinateur et son imprimante.

Allons plus en détail dans le fonctionnement d'un protocole par un exemple courant : l'email A chaque envoi, l'e-mail va être d'abord transmis au serveur de l'émetteur qui, à son tour, va transmettre le message au serveur du destinataire. Dans l'exemple ci-dessous se trouvent les messages typiques envoyés entre un client (en vert) et un serveur (en rouge) à travers le protocole SMTP (cf.Tableau 3 : Protocole Smtp [13]).. Toujours au début de « phrase », appelé **paquet**, se trouve une clé d'échanges permettant une compréhension du message envoyé entre le client et le serveur. Comme tout échange et discussion entre serveur et client civilisés, les premières phrases font guises de présentation (identification) et de salutation.

```
220 smtp.xxxx.xxxx SMTP Ready
HELO client
250 Hello client, pleased to meet you
MAIL FROM:<user@xxxx.xxxx>
250 <user@xxxx.xxxx> ... Sender ok
RCPT TO:<user2@yyyy.yyyy>
250 recipient ok.
DATA
354 Enter mail, end with "." on a line by itself
Subject: Test
250 Ok
QUIT
221 Closing connection
```

Tableau 3 : Protocole Smtp [13]

Utilisation dans le travail : Un Serveur central OntoNostra au milieu d'une architecture (cf. Figure 2 : architecture OntoNostra) qui comporte une série de petits clients (OntoMea) qui interagissent uniquement avec lui. OntoNostra est une instance de notre outil OntoMea qui, cependant, possède une base de données beaucoup plus robuste et ainsi que des web services utilisées par des clients pour envoyer et même recevoir des données et des métadonnées. Dans notre cas, outre les rôles principaux d'un serveur listé précédemment,

OntoNostra doit.

- Gérer la base de connaissance intégrée dans le serveur central
- Envoyer des requêtes aux autres serveurs. Dans 2 cas bien précis
  - Quand il ne réussit pas à répondre
  - Quand il sait de plus amples informations sur la donnée demandée.

OntoMea, machine cliente, peut :

- Recevoir des données (images, vidéo, documents…) ou bien des métadonnées (fichiers rdf), à travers des requêtes via web services, qu'OntoMea intègre à sa propre base de connaissance.
- Envoyer, à travers de requêtes via web services, que cette fois OntoNostra intègre à sa propre base de connaissance.
- Stocker des images sur le serveur pour les reprendre sur une autre instance OntoMea sur un autre poste.

Les clients n'ont pas encore le droit d'échanger des informations entre eux sans que ces informations ne passent pas auparavant par OntoNostra. Pour avoir une meilleure vision globale des scénarios possibles entre le client et le serveur avec ce type d'architecture, référez-vous au chapitre **Scénarios**. De suite, se trouve un petit résumé des points forts et faibles que cette architecture peut apporter.

Points forts :

- Simplicité de fonctionnement.
- Disponibilité 24/24h des informations mises à disposition

Points faibles :

- Besoin d'une machine avec une puissance toujours plus grande et gourmande en processus au fur et à mesure de l'augmentation des clients.
- Si le serveur central tombe tout casse, par conséquence le serveur est le point faible du réseau

4.2 P2P

Un second type d'architecture web intéressante à considérer pourrait se reposer sur un fonctionnement semblable au P2P *(cf. Figure 10 : Client/serveur vs P2P– schéma de droite)***.** On entend souvent parler de p2p, néanmoins comment fonctionne-t-il ?

Le Peer to Peer, Paire à paire, est un modèle de communication Web entre machines égales ayant accès à internet toujours à travers un protocole, ici HTTP. Ce qui veut dire que les machines qui appartiennent à ce type de réseau ont les deux rôles principaux du web une fois clients et une autre fois serveur. L'objectif du P2P est de relier tous les possesseurs de la même information. Par conséquence, c'est le moyen le plus simple pour partager les ressources personnelles (image, vidéo, musique...).

Fonctionnement : Une fois que le demandeur trouve l'information désiré les deux machines peuvent s'échanger l'information sans passer par une machine tierce et par conséquence perdre du temps. **Les exemples** de programmes utilisant cette technique sont nombreux : eMule, Limewire... Faite attention, ce n'est pas l'architecture P2P qui est interdite, c'est l'échanges de ressources et d'informations duquel l'utilisateur n'a pas le droit d'auteur.

Utilisation dans le travail : Les différents OntoMea pourraient, comme expliqué précédemment, prendre parfois :

- le rôle du **client** (demande des informations et reçoit ses informations sous la forme de métadonnées)
- Le rôle du **serveur** (cherche et envoie les métadonnées).

En effet, le serveur OntoNostra perdra de sa grande importance, sans perdre complètement de son utilité.

Points forts :

- Diminue la charge des serveurs.
- Les deux machines s'envoient des informations directement sans passer par des machines tierces. L'utilisateur peut choisir à qui donner les informations

Points faibles :

- Complique énormément la notion de recherche d'informations et l'information
- Les informations ne sont pas disponible 24/24h. Si machine éteinte pas possible d'avoir les données.

4.3 EVOLUTION POSSIBLE

Comme souvent dans le monde, la meilleure solution est un mixte entre les deux. Si le p2p se présente avantageux sur certain point, il ne faut cependant pas renoncer complètement à l'idée du serveur central, car il pourrait servir d'annuaire, de zone de stockage, de base pour les web services. Ainsi, il y aura une fusion des points forts de l'architecture client/serveur et P2P sans comme enlever tous les points faibles de ces architectures.

Par exemple (*cf. Figure 11 : Evolution p2p [14]*). : Un OntoMea recherche des informations sur un pays pour y partir en voyage. Il enverra alors une requête au serveur central qui lui cherchera si l'information se trouve sur le serveur central ou sur un autre OntoMea. Dès la découverte du OntoMea plus apte à envoyer les fichiers au format RDF, ils s'échangeront les informations ici les données sur ce pays.

- Le serveur OntoNostra aura le rôle d'intermédiaire et d'identificateur et aussi si l'information se trouve chez lui, d'envoyer les métadonnées via Web Service au format RDF/OWL
- Les deux OntoMea enverront une requête à OntoNostra et s'échangeront les fichiers directement via les Web Services offert par OntoNostra (sous format métadonnées ou même des images ou vidéos).

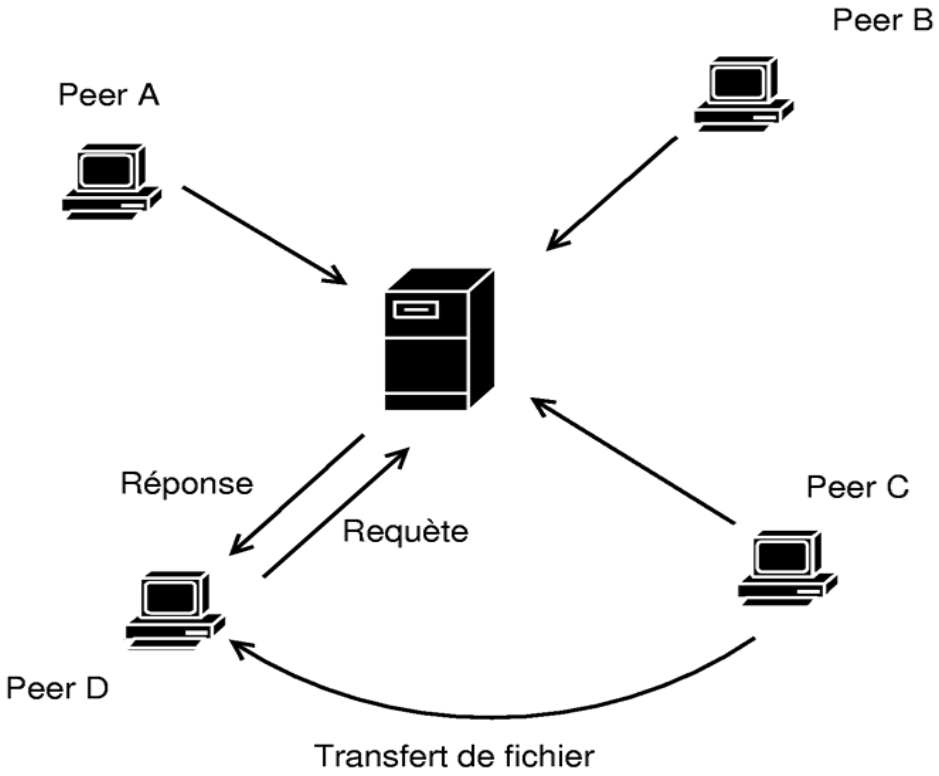


Figure 11 : Evolution p2p [14]

4.4 REST

Après avoir mis en place l'aspect du fonctionnement de l'architecture, je me suis intéressé à la mise en place d'un style d'architecture qui est a été crée récemment et qui cependant reprend des idées très anciennes, le REST. Il a été mis en évidence dans le monde Web par la thèse de **Roy T. Fielding** dans sa dissertation ***"an architecture style of networked systems"*** (traduction en annexe cf. 17.5). Fielding participe aux travaux du W3C et est co-fondateur du projet Apache.

"Representational State Transfer évoque l'image du fonctionnement d'une application Web bien construite : un réseau de pages Web (une machine à états finis virtuelle) où l'utilisateur progresse dans l'application en cliquant sur des liens (transition entre états) ce qui provoque l'affichage de la page suivante (représentant le nouvel état de l'application) à l'utilisateur qui peut alors l'exploiter" traduction d'une phrase de **Roy T. Fielding**

REpresentational State Transfer n'est pas un protocole, un standard ou encore un format. Il est un **style d'architecture orienté ressource (ROA)** pour des systèmes distribués. Un style d'architecture est un ensemble de contraintes qui permettent, lorsqu'elles sont appliquées aux composants d'une architecture, d'optimiser certains critères propres à la solution à concevoir.

REST reprend l'architecture originale du World Wide Web et de l'http (HyperText Transfer Protocol). L'http est un protocole de communication entre les machines que le Web à mis en place au tout début du Web, une ample explication est présente au chapitre suivant. Toute chose assez importante pour être référencée peut être ressource : un article, une photo ou bien un service…. Une ressource peut par conséquence être un objet physique ou un concept abstrait. Ces ressources sont accessibles via un URL (Uniform Ressource Locator) ou un URI (Uniform Ressource Identifier) qui sont très importants dans le problème d'identification des ressources et qui va être abordé au chapitre suivant. La notion d'architecture orientée ressource est un point vraiment très important aussi dans le Web Sémantique et les fichiers RDF.

Le REST affirme que tout se trouve déjà dans l'http. **L'HTTP,** un protocole sans état (stateless) ce qui signifie que les requêtes précédentes ou suivantes ne sont pas enregistrées. Ce qui permet de donner une réponse à une requête directement sans aller chercher des informations ultérieures sur cette requête. Chaque nouvelle page visualisée contient toutes les informations nécessaires pour afficher la ressource appropriée ou effectuer les traitements demandés par le client. HTTP a un nombre très limité d'opérations cependant suffisante à la plupart des travaux. Voici un tableau récapitulatif (*cf. Tableau 4 : Relation entre SQL et http*) présentant les différentes fonctions (nommés verbes) que peut avoir l'http et en parallèle 4 actions d'une base de données (Retrieve, Create, Update et Delete appelés **CRUD**) ainsi que l'action produite :

Action	SQL	http
Créer une nouvelle ressource	Create	**PUT**
Obtenir la représentation d'une ressource	Retrieve	**GET**
Mettre à jour une ressource	Update	**POST**
Effacer une ressource	Delete	**DELETE**

Tableau 4 : Relation entre SQL et http

Il existe, entre autres, deux autres verbes HEAD et OPTIONS qui peuvent être très utilisés dans le style d'architecture dédié à REST :

- **HEAD** : permet de récupérer uniquement les métadonnées (XML ou RDF dans le cas du travail)
- **OPTIONS** : permet de connaître les verbes autorisés pour une URI donnée.

Le web est, par conséquence, une gigantesque collection de ressources qui peuvent servir aux utilisateurs et aux machines de différentes manières, et dont l'affichage par page HTML est juste l'une d'entre elles. Cette ressource peut, donc, aussi bien être un PDF, un JPG et dans notre projet un fichier de métadonnées .RDF.

Le choix de Rest a tout de même du être précédé par la contraposition d'autres types et de style d'architecture dont la plus connues est SOAP avec la création de cookies et de sessions. **RPC** (Remote Procedure Call) et son évolution **SOAP** acronyme de Simple Object Access Protocol sont très différents dans l'esprit à REST car ils ajoutent une couche en dessus du Web. SOAP est un des protocoles définit par le W3C, basé sur XML, pour l'échange de messages entre différents composants software. Il s'agit d'une évolution du RPC puisqu'il est plus sûr en lui rajoutant une couche. Trois raisons principale ont influencé mon choix et m'ont poussé à ne pas le choisir :

- REST s'adapte mieux aux besoins du travail diplôme
- SOAP a une tendance à être très vite « lourd ». Un service SOAP se met en place comme le déploiement d'un objet et par la suite de la modélisation
- Deux services faits avec du SOAP peuvent être entre eux non interopérables

Points forts :

- Facilité d'utilisation, seule la compréhension d'http est nécessaire pour le fonctionnement de REST.
- Etant une architecture dédiée aux ressources **(ROA),** elle s'adapte très bien au Web Sémantique.
- Gain en interopérabilité entre client et serveur, elle n'ajoute pas de couche par-dessus.
- De plus en plus de sites web et applications connus (Amazon, eBay, Yahoo, Flickr…) proposent du Rest par conséquence donc une meilleure interopérabilité avec ces sites.

Points faibles :

- La confiance doit être un point important pour l'identification des ressources.
- La ressource n'a pas la possibilité d'être déplacé, mais peut être modifié
- Architecture nouvelle donc pas beaucoup de document mis à disposition à travers le web

Utilisation dans le travail : le projet se base sur le Web Sémantique ainsi que sur l'échange et l'identification de ressources. En tenant compte que Rest est aussi une architecture orientée ressource, nous pouvons donc remarquer le parallèle entre les deux. Rest n'est pas une couche par dessus le Web, c'est le Web actuel, passé et futures. Il n'y aura jamais de problèmes d'interopérabilité entre un client OntoMea et le serveur OntoNostra car tout se passe à travers le protocole dédié à internet : http (GET, POST, HEAD…).

Résultat, le passage d'un type d'architecture de client/serveur à P2P avec un serveur centrale (utilisé pour des services spécifique) sera d'autant plus facilité (cf. Section 4.3). Tout cela se fera de façon transparente pour les utilisateurs, sans qu'il ne fasse aucun changement.

4.4.1. HTTP DEFINITION CODE

L'http envoie des codes de statut pour savoir que faire ensuite entre le client et le serveur. Il paraît important à mes yeux de présenter une liste regroupant les codes les plus importants, vu l'importance qu'ils auront tout au long de ce document :

Code	Message	Signification
1xx	**Information**	
100	*Continue*	Attente de la suite de la requête
101	*Switching Protocols*	Acceptation du changement de protocole
102	*Processing*	Traitement en cours (évite que le client dépasse le temps d'attente limite).
2xx	**Succès**	
200	*OK*	Requête traitée avec succès
201	*Created*	Requête traitée avec succès avec création d'un document
202	*Accepted*	Requête traitée mais sans garantie de résultat
3xx	**Redirection**	
300	*Multiple Choices*	L'URI demandée se rapporte à plusieurs ressources
303	*See Other*	La réponse à cette requête est ailleurs
4xx	**Erreur du client**	
400	*Bad Request*	La syntaxe de la requête est erronée
401	*Unauthorized*	Accès à la ressource refusé
403	*Forbidden*	Refus de traitement de la requête
404	*Not Found*	Document non trouvé
405	*Method Not Allowed*	Méthode de requête non autorisée
5xx	**Erreur du serveur**	
500	*Internal Server Error*	Erreur interne du serveur
501	*Not Implemented*	Fonctionnalité réclamée non supportée par le serveur
503	*Service Unavailable*	Service indisponible

Tableau 5 : Liste code [16]

5. IDENTIFICATION

Poursuivant le travail par le développement de l'idée d'Identification. Qui est qui ou quoi est quoi ? L'identification dans Rest est un point central car, sans savoir avec qui l'information est échangée, il est difficile de savoir quoi faire ainsi que de trouver la **ressource**. L'identification dans le web sémantique est également un point très important puisque sans celle-ci l'indexation, les comparaisons et le traitement des données typiques du web sémantique seraient impossibles.

L'identification, dans l'architecture proposée, se fera sur deux niveaux distincts, présentés dans les sections suivantes :

- Identification des OntoMea (machine cliente)
- Identification des Ressources par URI (Personne, images, lieux géographiques)

5.1 INDENTIFICATION DES SERVEURS ONTOMEAS

L'identification d'un OntoMea est différente de l'identification de son utilisateur! En effet un utilisateur peut avoir plusieurs OntoMea, par exemple un à la maison ainsi qu'un autre à son lieu de travail. L'identification de chaque OntoMea est cruciale pour la gestion des données. L'architecture globale se fait autour d'un serveur central OntoNostra. Cependant comme précisé au chapitre 4 elle évoluera semble très proche et évoluera vers une architecture P2P. Il a été important d'envisager une solution la plus souple possible, où l'on ne fait pas forcément de distinction (au niveau technique) entre le serveur central et un OntoMea local, chacun ayant sa propre URI/URL (son identification).

Le style d'architecture choisi est **Rest**, au niveau de la communication entre les OntoMea et le serveur central OntoNostra. Les OntoMea locaux vont être configurés pour connaître automatiquement l'adresse du serveur central. Il leur sera attribué un identifiant unique par OntoNostra avec une combinaison entre l'utilisateur et le nom attribué par exemple : **OntoMeaNicolaMaison** comme résultat l'URI aura cet aspect : http://OntoMeaNicolaMaison/ressources/123**.** Par la suite grâce à ce travail d'identification, il y aura la possibilité de choisir avec qui échanger des informations ce qui signifie évoluer vers du P2P.

Pour reconnaître les serveurs, j'ai choisi d'utiliser une carte d'identité nommée (certificat) qui peut identifier le serveur. Ce certificat va donc être utilisé de manières différentes dans le but d'authentifier et identifier l'OntoMea. De plus amples informations sur le certificat et son mécanisme se trouvent dans le chapitre sur la **sécurité**.

5.2 INDENTIFICATION DES DONNEES – L'URI

Un des points les plus importants dans le Web sémantique est l'identification unique de la ressource sans laquelle il serait impossible de partager l'information. Un exemple concret est Michael Schumacher : sans un identifiant, comment peut-on diversifier ce pilote de formul1 à Michael Schumacher ce bon vieux cordonnier ? Cette dernière vit en Allemagne et reçoit des milliers de lettres de fans sans savoir pourquoi. Pour l'identifier, nous allons utiliser un URI (Uniform Ressource Identifier) qui comme son nom l'indique sert à identifier un concept, un objet. L'URI peut être, l'exemple le plus connu, l'URL qui constitue les « adresses des pages Web », l'Uri fonctionne comme une clé primaire dans une base de données.

Voici un exemple issu d'un site *[17]* qui permettra d'expliquer d'une meilleure façon le fonctionnement d'un URI : *"En cliquant sur l'URI suivante dans cette exemple un URL* ***http://fr.weather.com/weather/local/FRXX0076****, vous obtenez la météo d'Oaxaca une ville du Canada. Le fait de cliquer sur le lien indique au navigateur que vous voulez obtenir une représentation de la ressource désignée. Le navigateur reconnaît http comme protocole et envoie alors au serveur fr.weather.com une requête HTTP GET sur le port 80. En retour, le serveur lui envoie le code* ***200*** *(ok) et la représentation actuelle de la ressource /exemple.com/Oaxaca. Cette représentation comporte deux parties, des méta données qui décrivent le contenu de la représentation et l'URI qui l'identifie." [17]*

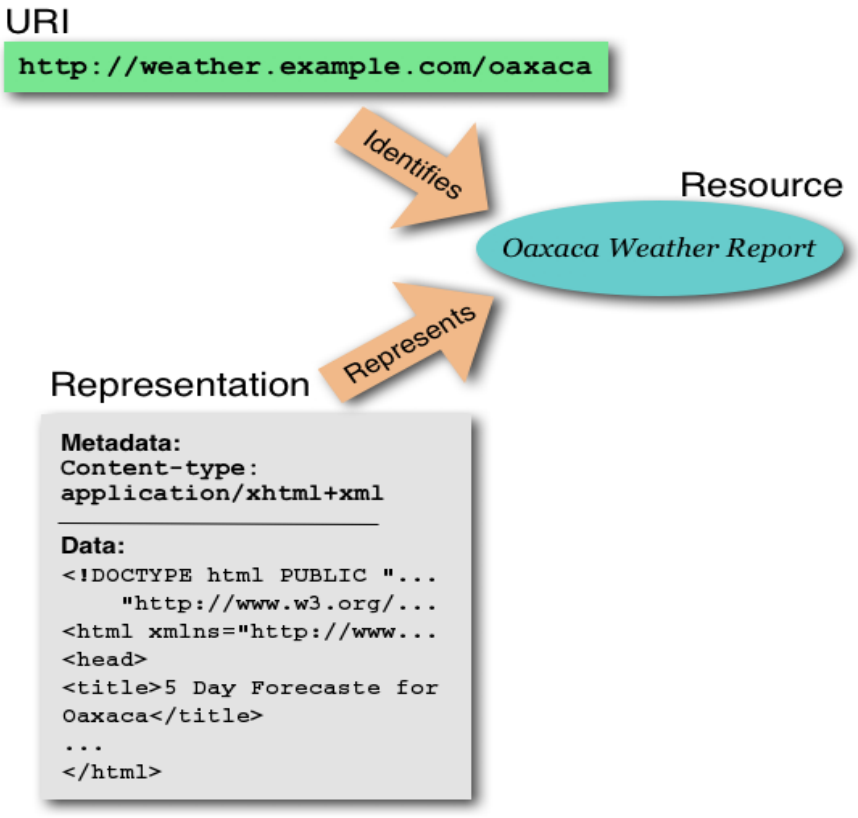


Figure 12 : ressource URI représentation [18]

Pour l'identification, nous avons repris donc la même idée que l'architecture REST (Representiational state transfer). L'un des points de Rest, expliqué au chapitre précédent, est qu'une URI doit être descriptive. Par exemple, en lisant cette adresse Web : http://www.blog.org/People/nicola/2008/04/02/commentaireJean, on comprend qu'il s'agit de la page Web d'une personne qui s'appelle Nicola, écrite le 2 avril 2008 avec des commentaires de Jean. Un document très bien écrit de W3C explique l'aspect que doit avoir un URI dans le web sémantique. Le document en question, dont je vais reprendre les grandes idées, sera présent en annexe du document **voir section 17.4 - Cooluris**.

Connaître l'URI doit suffire pour accéder à une ressource. **Par exemple**, quand on va sur une page Web, le navigateur fait un http GET de l'URL tapée et en retour on obtient une page Web. Les pages Web ont habituellement des images. Celles-ci sont des ressources séparées qui possèdent leur propre URI. La page Web donne uniquement les URLs vers les images et le navigateur fait des HTTP GETs supplémentaires sur elles jusqu'à ce que toutes les ressources soient obtenues et la page complètement affichée. L'aspect important est que des noms communs de natures très différentes (une image, un texte, une vidéo, un mp3, un diaporama...) peuvent être traités de la même façon. Je peux donc tous les prendre de la même façon avec une URL grâce à un simple GET.

La définition des URI ne peut être la conséquence d'un développement. Les URI doivent être spécifiés au moment de la conception. Les URL actuels comportent des extensions qui dépendent d'une technologie comme .aspx, .html ou PHP. Notre solution doit représenter les URI de manière indépendante d'une technologie. Par exemple, nous obtiendrons à travers l'URL www.ontonostra.ch/files/images/12345 et grâce à une requête http GET, la représentation de l'image 12345. Cette URL est indépendante de toute technologie. Cette image aurait pu être un .GIF, un .JPG ou bien un .bmp. Pour que le serveur web sache s'il travaille avec un fichier .doc ou un ficher .pdf il est nécessaire de paramétriser la négociation du contenu.

Pour se faire, la plupart des grandes plateformes serveur Web comme apache ou IIS, ont mis en place une fonction d'**UrlRewriter**. Comment fonctionne-t-il ? Il utilise une fonction de réécriture configurable par règles (construit à partir d'un interpréteur d'expressions régulières) pour réécrire au vol des URL et pour leur donner un aspect plus « **sexy, cool** », plus compréhensible. L'UrlRewriter fonctionne aussi pour un autre point important des URI, gérer les URIs de façon à ce qu'elles soient toujours valides dans 2, 20, ou 200 ou même 2000 ans

Par exemple : Grâce à la fonction d'UrlRewriter, une adresse URI de ce type : http://www.votresite.tld/article.php?numero=8125&page=2 pourrait facilement devenir http://www.votresite.tld/article-8125-2. On remarque tout de suite ce qu'UrlRewriter peut apporter à Rest et au travail d'identification sur nos URI. Le site http://del.icio.us/ de gestion et de partage de favoris est un exemple de site construit avec des URI logique qui servent de base directe au système de navigation et de recherche du site.

Au niveau du Travail de diplôme, chaque donnée décrite en RDF doit avoir son URI: pour l'instant nous gérons au minimum des personnes avec des fichiers FOAF, des images avec RDF, des lieux géographiques. Pour nos travaux, le nom de domaine utilisé est celui de notre site internet sur le web sémantique et ses outils : www.websemantique.ch. Selon le modèle proposé par le document **'cool URIs'** de W3C (17.4 Cooluris), la description d'une ressource demande 3 URI :

- **Resource Identifier** : URI pour identifier la ressource, il s'agit de l'URI utilisé dans les triplets http://www.websemantique.ch/typeRessource/IDRessource. Le type Ressource étant 'people', 'images', etc.
- **HTML document URL** : un URI pour identifier les données destinées à l'humain http://www.websemantique.ch/typeRessource/doc/IDRessource ou le nom de fichier. L'extension n'est pas spécifiée, il peut s'agir d'un .html pour une page de description, d'un .JPG pour une image, etc.
- **RDF document URL** : un URI pour identifier les données RDF, destinées aux machines http://www.websemantique.ch/typeRessource/RDF/IDRessource, il peut s'agir de RDF/XML, de n3, etc.

Une des solutions pour gérer ceci dans le web sémantique est la suivante (cf. Figure 13 : Echange d'information Client/serveur [18]) : une requête du client se fait sur le serveur par l'URI de la ressource, et en fonction du format de réponse attendu (html ou RDF), le serveur renvoie un code http **303 redirect** sur l'URI qui convient. A ce moment le client grâce à GET sait le contenu de la réponse et peut l'accepter ou non et renvoie **200** si ok.

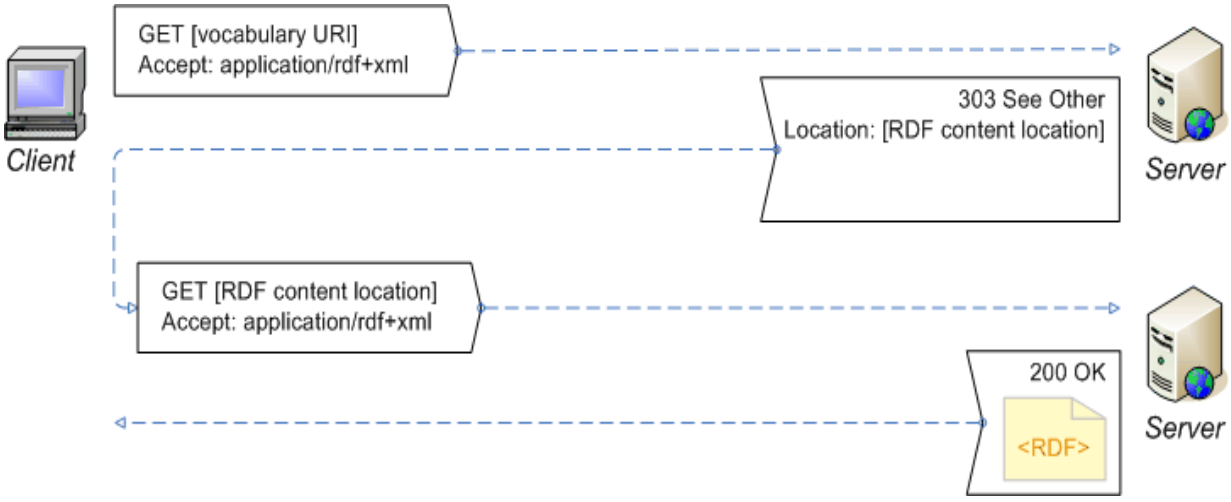


Figure 13 : Echange d'information Client/serveur [18]

OntoNostra donne la possibilité d'héberger des fichiers, par exemple des images d'un voyage qui ne sont pas des métadonnées. Pour ce qui en est de pages html il pourrait, par exemple, afficher une page .html de présentation d'une personne en affichant simplement les données foaf ou choisir d'autres pages voulues par l'utilisateur.

Les sous-sections suivantes présentent notre solution pour traiter les exemples des personnes, des fichiers/images et des lieux.

5.2.1. PERSONNES

Le problème devient de plus en plus aigu quand il s'agit de se désigner. Je suis une personne, un humain de sexe masculin allant sur sa 25ème année. Ceci constitue une partie de ma réalité physique. J'ai une réalité sexuelle, une réalité professionnelle, une réalité individuelle, etc. Le problème commence dès qu'il s'agit de désigner toutes ces réalités par une URI unique. Pas clair ? Prenons quelques exemples. Nicola DePalma est :

- **`nicola@gmail.com`** ou alors est-ce l'adresse courriel personnelle de moi-même Nicola DePalma qui permet de lui envoyer des courriels privés?
- **`nicoladepalma@students.hevs.ch`** ou alors est-ce l'adresse courriel professionnelle Nicola DePalma qui permet de lui envoyer des courriels professionnels?
- **`http://www.pagepersoNicola.ch/`** ou alors est-ce l'adresse du site Web personnel?
- **`http://www.travaille.org/People/nicola/`** ou alors est-ce l'adresse de la page professionnelle de Nicola DePalma?
- **`urn:nicola-depalma`** ou bien est-ce une URN créée pour gérer des documents en local et qui se réfèrent à moi-même

Utilisation dans le travail : Pour répondre à cette problématique, on a choisi de gérer, les URI des personnes grâce à OpenID : un système d'authentification décentralisé qui permet le partage d'attributs (OpenID est décrit plus en détail à la section 6.1).

Les personnes sont décrites avec un fichier FOAF qui lui-même possède une propriété OpenID **<FOAF : OpenID/>**. Dans la mesure du possible, OntoNostra sera capable d'attribuer un OpenID à un nouvel utilisateur. Dans le cas contraire, nous utiliserons des OpenID existants que nous pouvons déjà obtenir en s'inscrivant sur divers sites. Si OntoNostra attribue un OpenID, la forme idéale sera: http://www.websemantique.ch/people/alice

Les 3 URIs auront donc la forme suivante:

- http://www.websemantique.ch/people/alice: **Identificateur** de la personne voir aussi OpenID section 6.1
- http://www.websemantique.ch/people/doc/alice: **Home page** de la personne, par exemple données FOAF au format HTML
- http://www.websemantique.ch/people/RDF/alice: **RDF (FOAF)** de description de la personne fichier FOAF (cf. Figure 6 : Exemple mon fichier FOAF)

5.2.2. FICHIERS/IMAGES

OntoMea organisant les fichiers personnels doit leur attribuer un URI pour l'identifier. Un même fichier peut avoir plusieurs chemins locaux ou/et réseau (il peut se trouver en local, sur Flickr, sur OntoNostra, etc.). Pour gérer tout cela une ontologie memOnto a été mise en place. Cette ontologie reprend tous les chemins qui mènent au fichier spécifié. Afin de les reconnaître, les fichiers (images, documents, vidéo, etc.) obtiennent un identifiant unique par un checksum sha-sum qu'OntoMea gère déjà dans ses attributs.

Définition checksum

*« Le **checksum** d'un fichier est une séquence de chiffres et de lettres définissant de manière précise (néanmoins non unique) un fichier afin de savoir s'il a été altéré »* [20] et dans notre cas aussi pour **l'identifier**, par ce fait deux rôles, un de sécurité et l'autre d'identification.

La fonction de checksum qui est déjà mis en place dans OntoMea, a un algorithme très évoluée. La fonction contrôle le fichier bits par bits il sera donc très difficile, cependant non impossible, d'avoir deux fois le même checksum pour des fichiers différents. Il existe, donc par conséquence, un risque très faible d'avoir le même URI pour deux ressources différentes.

Utilisation dans le travail : Maintenant un exemple de résultat d'un checksum (*cf. Figure 14 : Exemple checksum*) obtenu grâce à OntoMea reçu sur un fichier quelconque du type image. Ce même fichier aura le même checksum s'il se trouve en local, sur le Web ou encore sur un autre ordinateur :

```
9fd5eda5847c45f0df84a973f6412b03305eefdef400e6dec634711f8c6bab08
```

Figure 14 : Exemple checksum

Les 3 URIs pour les fichiers dédiés aux images, fonctionnant exactement comme les fichiers dédiés aux personnes, seront ainsi attribués de la manière suivante et utilisés dans la base de connaissance :

- http://www.websemantique.ch/images/123456: **Identifie** une image, 123456 est aussi le checksum donné pour ce fichier.
- http://www.websemantique.ch/images/doc/laPhoto Il y a deux possibilités de création d'URI : on peut mettre le nom du fichier ou le checksum du fichier comme identifiant sans son attribut/type de fichier ici .jpg http://www.websemantique.ch/images/doc/123456 : le **fichier .jpg** de la photo en question.
- http://www.websemantique.ch/images/RDF/123456: **document RDF** avec la description de la photo.

5.2.3. LIEUX - GEONAMES

Pour l'aspect géographique (ville, pays, voyage…) OntoMea a intégré **GeoNames**. Il n'y a pas, par conséquence, à gérer d'URI de pays mais à les récupérer dans la base de connaissance. GeoNames est une base de données Sémantique et gratuite dédié à la géographie et accessible par internet. GeoNames offre des Web Services disponible pour tous et qui renvoient les informations sur les différents lieux recherchés. Ce projet fonctionne come un Wiki ce qui veut dire que tout les utilisateurs sont libres d'ajouter ou améliorer les données.

Chaque identifiant (lieu) possède des informations comme la latitude, la longitude, l'altitude, la population, le code postal, la région ou le canton, le pays tout cela en différentes langues. En plus de ces informations, un lien vers les articles de Wikipedia consacrés à ce lieux y sont ajoutée Il y a deux possibilités de l'utiliser en se téléchargeant la base de donnée directement mais manque de mise à jour ou l'utilisation des Web Services proposés par GeoNames. Il est directement intégré au Web Sémantique parce qu'implémenté par:

- Possède son propre dialecte **RDF** avec une Ontologie définie qui, par conséquence, intègre les informations entre elles (métadonnées).
- Possède un **URI** identifiant et stable correspondant chaque fois un fichier RDF. Les informations peuvent donc évolué sans que son URI y soit affecté.

Maintenant présenté ci dessous, le fichier XML *(cf. Figure 15 – Ville de Zermatt*) reçu par GeoNames via ses Web Services. Après avoir fait une requête sur un pays ou sur une ville, l'utilisateur peut utiliser directement les données acquises. Il reçoit donc un simple XML avec toutes les informations listées précédemment qui seront donc très facile à utiliser pour une simple application.

```
- <geonames>
<totalResultsCount>4087</totalResultsCount>
- <geoname>
    <name>Zermatt</name>
    <lat>46.0166667</lat>
    <lng>7.75</lng>
    <geonameId>2657928</geonameId>
    <countryCode>CH</countryCode>
    <fcl>P</fcl>
    <fcode>PPL</fcode>
</geoname>
```

Figure 15 : exemple du résultat pour la ville de Zermatt

Utilisation dans le travail : OntoMea insère directement dans sa base de connaissance, les URI attribués par le projet GeoNames. Par exemple, si un utilisateur inscrit Martigny comme ville résidente l'inscrire dans le fichier FOAF (cf. *Figure 16 : Inscription de GeoNames dans Foaf*), OntoMea va inscrire uniquement **l'URI** de GeoNames, ici dans cette exemple : http://sws.geonames.org/2659748/ et non sa valeur littérale, son nom « **Martigny** ».

```
    <memo:isNowInLocation
RDF:resource="http://sws.geonames.org/2659748/"/>
    <memo:isOfNationality
```

Figure 16 : Inscription de GeoNames dans Foaf

Si vous cliquez directement sur l'URI dédié à Martigny, vous serez redirigés sur une page HTML se trouvant sur le site de GeoNames dédié à cette ville. La page (*cf. Figure 18 : about RDF de Martigny fait par* GeoNames) va contenir deux cartes géographiques avec aussi. Les informations listés précédemment et présentes également dans le fichier XML *(cf.* Figure 15 : exemple du résultat pour la ville de Zermatt). GeoNames donne aussi la possibilité de télécharger un fichier about.rdf consacré au lieu désiré.

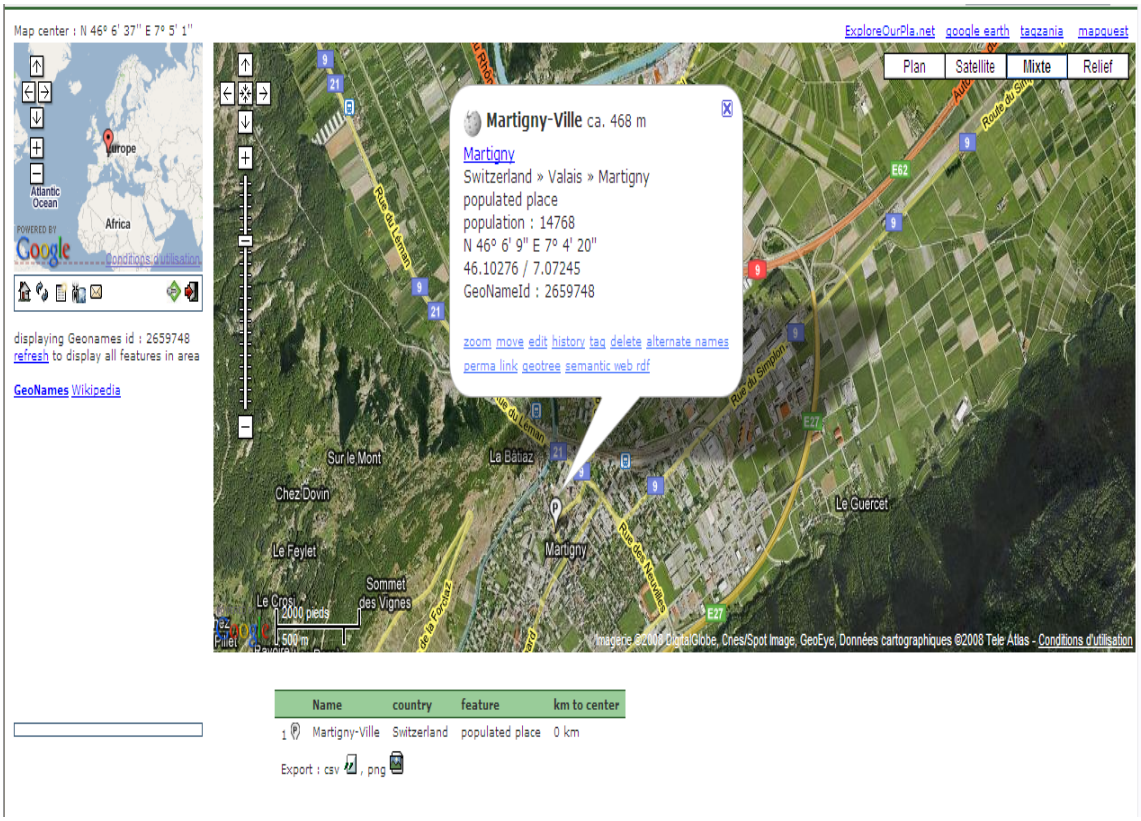


Figure 17 : Page Martigny GeoNames [21]

Au tour du fichier rdf (about.rdf) mis à disposition de GeoNames consacré à la ville de Martigny :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<rdf:RDF xmlns="http://www.geonames.org/ontology#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:wgs84_pos="http://www.w3.org/2003/01/geo/wgs84_pos#">
<Feature rdf:about="http://sws.geonames.org/2659748/">
<name>Martigny-Ville</name>
<alternateName>Martigny</alternateName>
<featureClass rdf:resource="http://www.geonames.org/ontology#P"/>
<featureCode rdf:resource="http://www.geonames.org/ontology#P.PPL"/>
<inCountry rdf:resource="http://www.geonames.org/countries/#CH"/>
<population>14768</population>
<wgs84_pos:lat>46.1027565693105</wgs84_pos:lat>
<wgs84_pos:long>7.07244873046875</wgs84_pos:long>
<parentFeature rdf:resource="http://sws.geonames.org/2659752/"/>
<nearbyFeatures rdf:resource="http://sws.geonames.org/2659748/nearby.rdf"/>
<locationMap>http://www.geonames.org/2659748/martigny-ville.html</locationMap>
<wikipediaArticle rdf:resource="http://fr.wikipedia.org/wiki/Martigny"/>
<wikipediaArticle rdf:resource="http://de.wikipedia.org/wiki/Martigny_VS"/>
<wikipediaArticle
rdf:resource="http://it.wikipedia.org/wiki/Martigny_%28Vallese%29"/>
<wikipediaArticle
rdf:resource="http://en.wikipedia.org/wiki/Martigny%2C_Switzerland"/>
<owl:sameAs
rdf:resource="http://dbpedia.org/resource/Martigny%2C_Switzerland"/>
<wikipediaArticle rdf:resource="http://es.wikipedia.org/wiki/Martigny"/>
<wikipediaArticle rdf:resource="http://da.wikipedia.org/wiki/Martigny"/>
<wikipediaArticle rdf:resource="http://frp.wikipedia.org/wiki/Martegn%C3%A9"/>
<wikipediaArticle
rdf:resource="http://nl.wikipedia.org/wiki/Martigny_%28Zwitserland%29"/>
<wikipediaArticle rdf:resource="http://rm.wikipedia.org/wiki/Martigny"/>
<wikipediaArticle rdf:resource="http://sv.wikipedia.org/wiki/Martigny"/>
<wikipediaArticle
rdf:resource="http://vec.wikipedia.org/wiki/Martigny_%28Va%C5%82exe%29"/>
<wikipediaArticle rdf:resource="http://vo.wikipedia.org/wiki/Martigny"/>
</Feature>
</rdf:RDF>
```

Figure 18 : about RDF de Martigny fait par GeoNames

Les 3 URIs seront utilisé exactement comme dans les exemples précédents car GeoNames reprend cette même différentiation

- http://sws.geonames.org/2659748/: **Identificateur** URI du lieu géographique
- http://www.geonames.org/2659748/martigny-ville.html **Home page** des lieux sous format HTML (*cf. Figure 17 : Page Martigny GeoNames [21]*)
- http://sws.geonames.org/2659748/about.rdf : **Document RDF** (about.rdf) avec la description du lieu. (*cf. Figure 18 : about RDF de Martigny fait par GeoNames*)

6. AUTHENTIFICATION

L'authentification est une exigence pour les échanges d'informations. Si un utilisateur veut partager ses données avec tout le monde, il préférera choisir lui-même à qui faire confiance, et de ce fait il voudra avoir des garanties de sécurité. Ainsi, tout accès par service doit être authentifié pour identifier celui qui demande la ressource. Nous utilisons par conséquence l'identité sociale pour l'accès. La confiance au réseau doit être absolue.

Pour résoudre ce problème d'identification et authentification, nous avons la possibilité d'utiliser l'attribut OpenID [24]. L'outil Web 3.0 : **OpenID** est un moyen facile et sécurisée de communiquer votre identité en ligne. Nous verrons, dès la section suivante, l'utilisation pas par pas et le fonctionnement d'OpenID. Ci-dessous vous, trouvez un graphique qui explique l'esprit décentralisé d'OpenID et dans le même cas l'authentification.

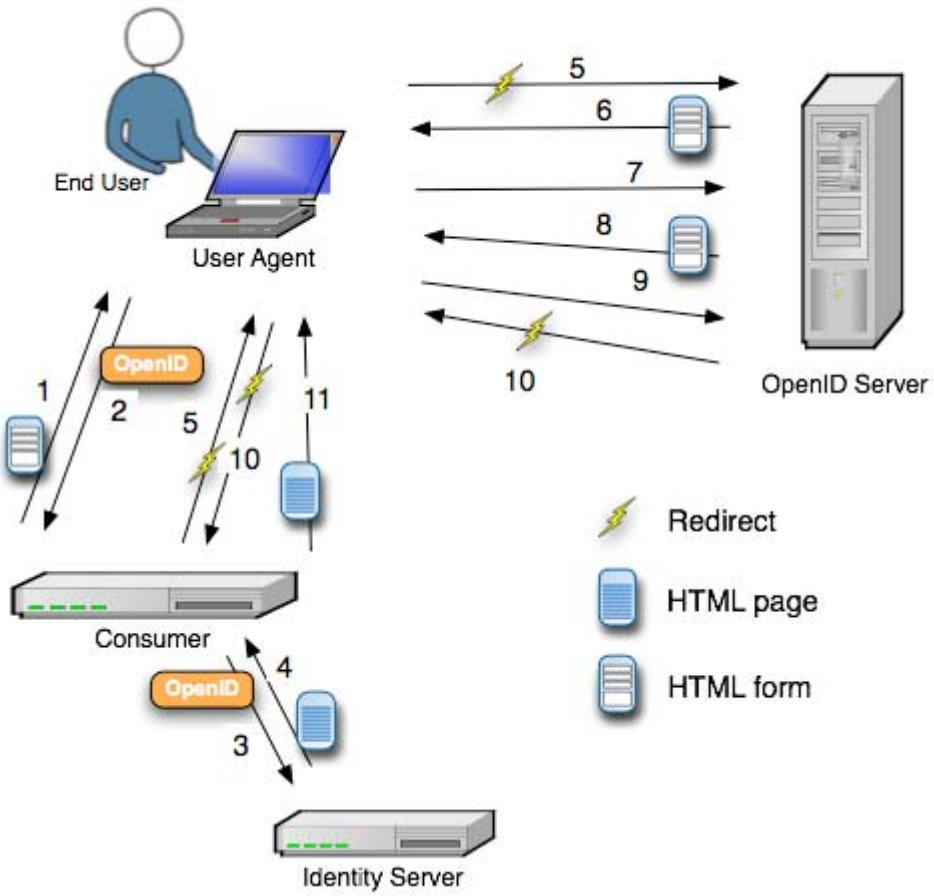


Figure 19 : Fonctionnement OpenID [29]

6.1 OPENID

OpenID permet à un utilisateur de retenir un seul login et un mot de passe pour tous les sites (de plus en plus nombreux : Yahoo, orange…) qui acceptent l'authentification par OpenID. Vous n'aurez à qu'à remplir vos données personnelles **non-obligatoires** comme nom, prénom ou encore l'email qu'une seule fois pour tous les sites avec OpenID. Cette authentification se base sur des liens de confiances établis entre les fournisseurs de service (sites Web utilisant OpenID par exemple) et les fournisseurs d'identité (OpenID providers ou Vertige).

La mise en place d'une identité OpenID est simple, il suffit de s'inscrire. Chaque personne peut maitriser la distribution de ses informations personnelles à un serveur OpenID (par exemple, MyOpenID). Une fois reçu un compte OpenID, on peut authentifier une URL comme nous appartenant en ajoutant deux métas – balises à sa page HTML principale : l'une pour pointer vers son adresse d'identité OpenID, l'autre pour pointer vers le serveur OpenID du fournisseur. Cependant, le fournisseur peut certifier que l'URL appartient bien au compte correspondant au site.

OpenID identifie et peut décrire les personnes FOAF. Une application évidente d'OpenID URI consiste à l'utiliser pour détecter les personnes dans un document FOAF (Il y a maintenant une propriété **foaf: OpenID** intégré par foaf). Dans le passé, il y a eu plusieurs discussions sur le meilleur format URI pour représenter une personne FOAF. Certains ont affirmé que cela devrait être la page d'accueil du site Web d'une personne tandis que d'autres ont soutenu que ce pourrait être une norme URI modèle de convention. OpenID pour l'URI d'une personne nous parait être une meilleure approche. Non seulement c'est une URI unique, mais elle a aussi un but fonctionnel - l'authentification des utilisateurs sur le Web. Pourquoi réinventer de nouvelles URI pour une personne si elle en possède déjà une qui l'identifie de manière unique dans de nombreux sites Web? Avec l'attribut de foaf : OpenID, nous pourrons rechercher une personne grâce à la requête SPARQL suivante :

```
Select ?who Where {    ?who foaf:openId <openid_uri>}
```

Figure 20 : Requête SPARQL

Pour l'élaboration de mon projet j'ai tout d'abord voulu, créer un fournisseur d'identité qui attribuerait un login du type www.depanico.websemantique.ch. Toutefois en y réfléchissant mieux et en voyant le temps que cela prenait, il est apparu que la meilleur solution dans le cadre de ce travail de diplôme est d'utiliser les serveurs qui attribuent déjà des OpenID. Ceci est en effet plus sûr. L'évolution d'OpenID va mener à la diminution des serveurs, de 2 ou 3. Pour l'instant, le fournisseur préalablement développé n'est pas fonctionnel par manque de temps. Nous ferons donc un exemple sur l'un des serveurs d'identification le plus grand www.myOpenID.net.

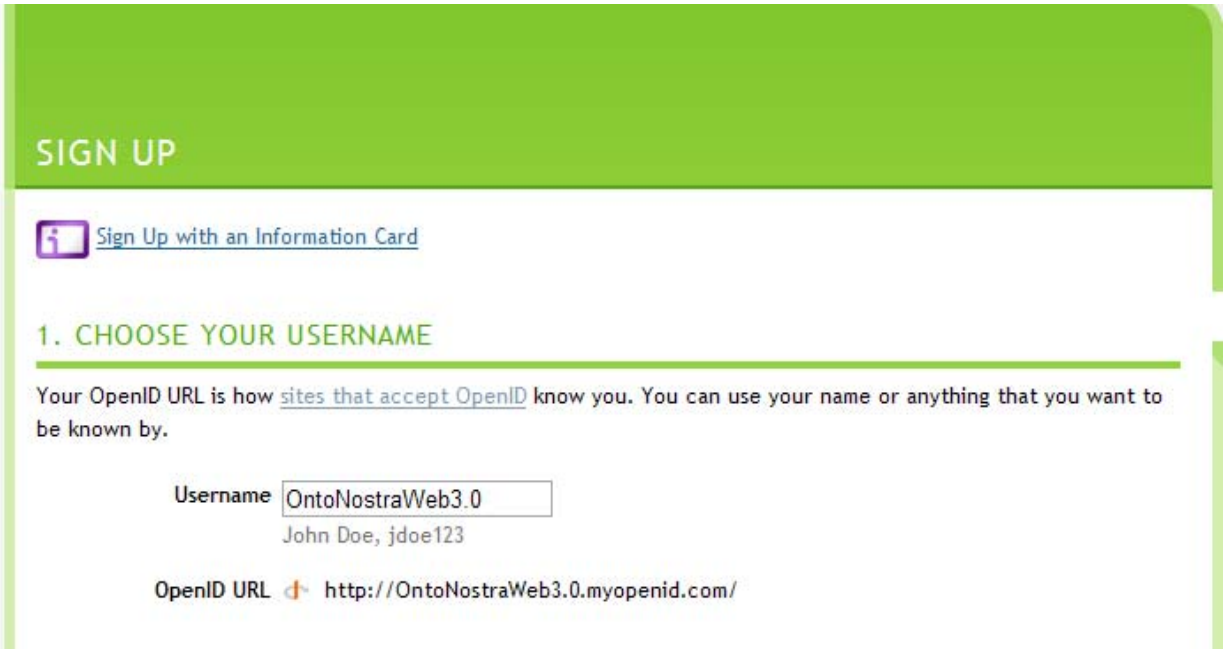
6.1.1. EXEMPLES D'UTILISATION D'OPENID :

Etape 1 : Enregistrement client.

Avant d'utiliser les services proposé par OpenID, l'utilisateur doit s'enregistre sur un fournisseur d'identité, dans le cas présent un des deux les plus connus MyOpenID (http://www.myopenid.net), l'autre étant Vertigo. Sur la page servant à enregistrer un utilisateur, il y a quatre étapes à remplir pour créer un OpenID (*cf. Figure 21 : Pages d'inscription OpenID partie 1 [26]*).

1. Choisir un nom utilisateur : **OntoNostraWeb3.0**) qui donnera ensuite un OpenID URL de ce type http://OntoNostraWeb3.0.myopenid.com/
2. Entrer un mot de passe.
3. Entrer e-mail (non obligatoire)
4. Ecrire le test qui se trouve dans le carré et accepter les termes des services proposés par OpenID

Dès lors il crée son compte. Celui-ci aura pour identifiant OpenID et page d'accueil : http://OntoNostraWeb3.0.myopenid.com/.



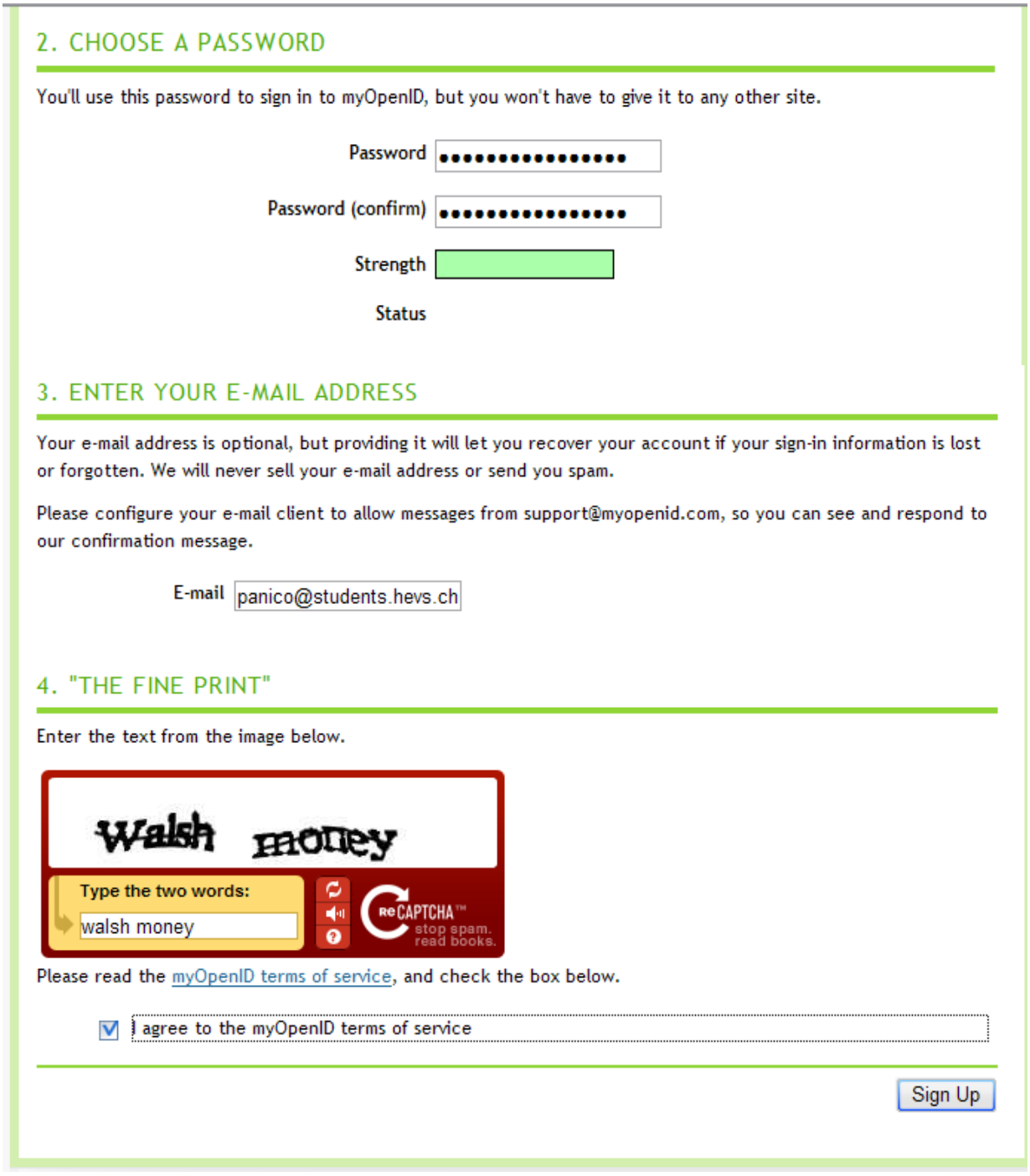
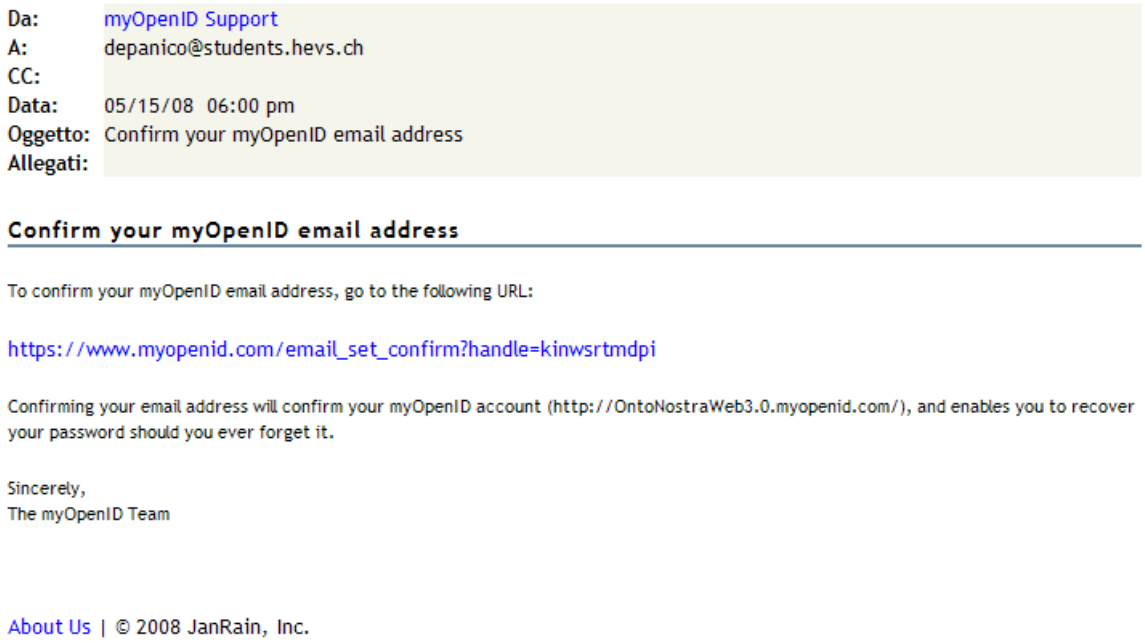


Figure 21 : Pages d'inscription OpenID partie 1 [26]

Etape 2 : Confirmation par email

Si l'utilisateur écrit son email (ce qui n'est pas obligatoire) il reçoit, comme d'habitude, un mail de confirmation (*cf. Figure 22 : Email OpenID Confirmation*) afin d'avoir la possibilité de le récupérer dans le cas où il l'oublierait.



Da: myOpenID Support
A: depanico@students.hevs.ch
CC:
Data: 05/15/08 06:00 pm
Oggetto: Confirm your myOpenID email address
Allegati:

Confirm your myOpenID email address

To confirm your myOpenID email address, go to the following URL:

https://www.myopenid.com/email_set_confirm?handle=kinwsrtmdpi

Confirming your email address will confirm your myOpenID account (http://OntoNostraWeb3.0.myopenid.com/), and enables you to recover your password should you ever forget it.

Sincerely,
The myOpenID Team

About Us | © 2008 JanRain, Inc.

Figure 22 : Email OpenID Confirmation

Etape 3 : Découverte menu et utilisation OpenID

Si l'utilisateur veut ajouter des données et avoir la possibilité de configurer son compte comme il le désire, il pourra se loguer grâce à son login et mot de passe. Voici de côté, le menu avec les différents choix d'utilisations auxquels l'utilisateur peut accéder.

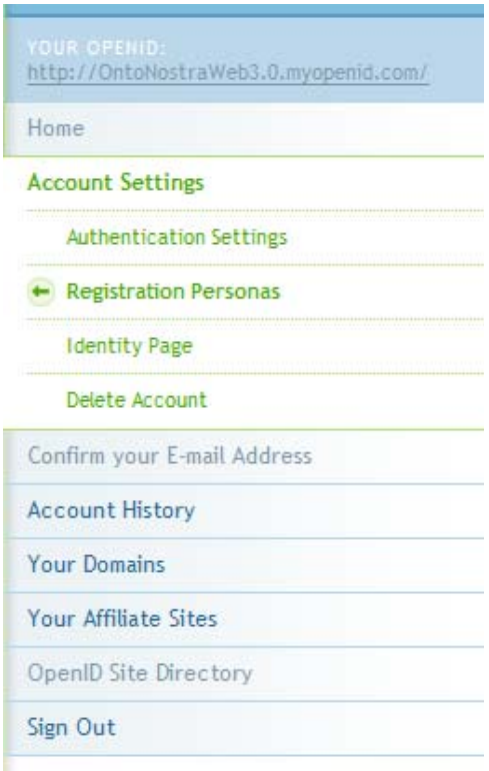


Figure 23 : Menu OpenID

Etape 4 : Page d'accueil personnel d'OpenID

En tapant dans son browser internet, son login URI attribué par **OpenID**, dans notre exemple, http://OntoNostraWeb3.0.myopenid.com/. On reçoit au retour une page de présentation personnelle (*cf. Figure 24 : Page d'accueil OpenID personnel*) mis à disposition au monde web par OpenID. L'utilisateur a donc la possibilité d'insérer ou non ou de modifier à tout moment les informations mis à disposition à tout moment.

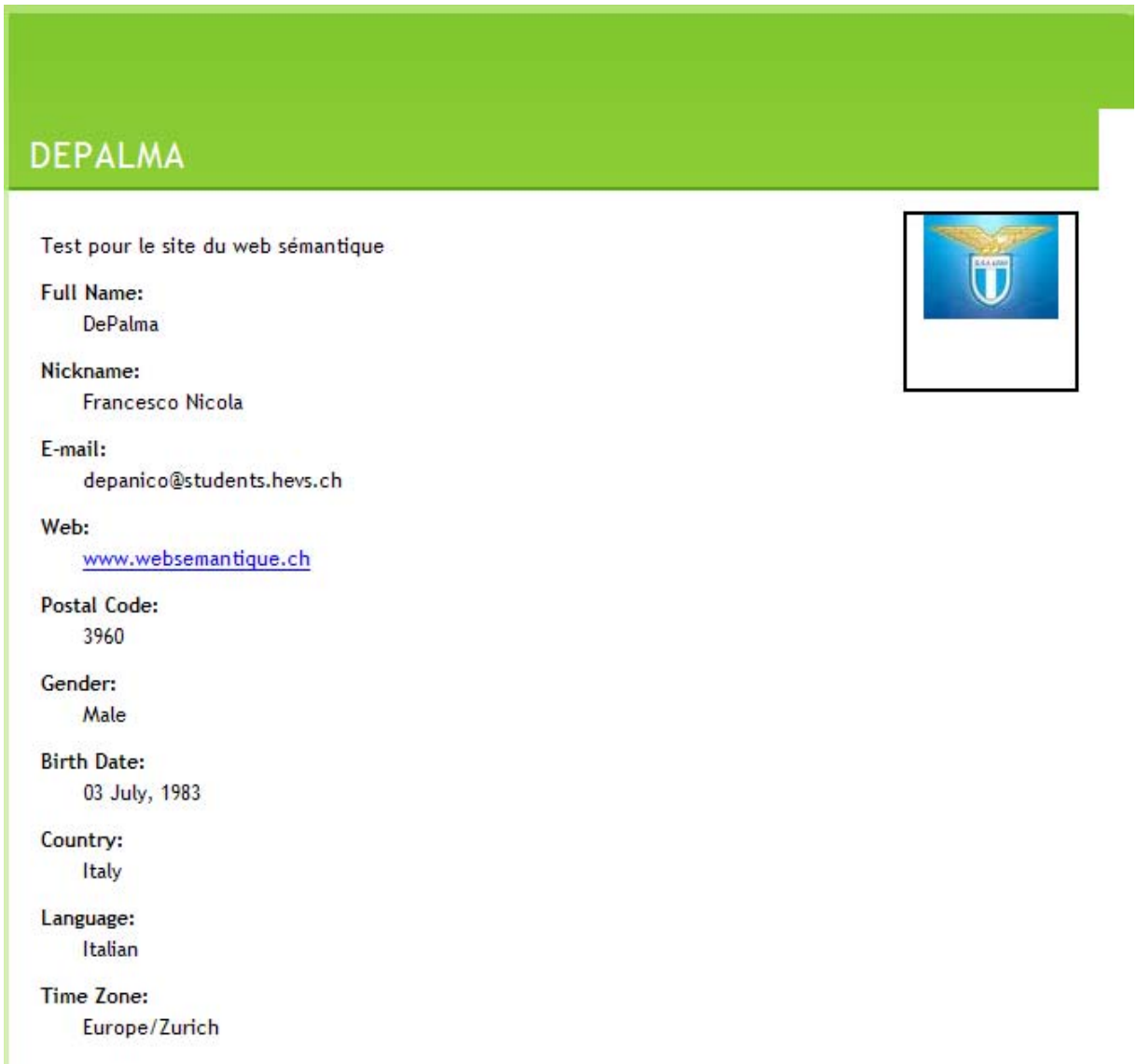


Figure 24 : Page d'accueil OpenID personnel

Etape 5 : Création d'identité

Dans cette page, l'utilisateur insère les informations personnelles qu'il veut mettre à disposition du publique et des sites utilisant le protocole OpenID. Il s'agit des mêmes informations que la plupart des sites demandent lors d'une inscription. La grande force d'OpenID est que l'utilisateur a le droit d'ajouter autant d'identités qu'il le souhaite dans son OpenID.

L'utilisateur sera libre, par conséquence, de mettre des données fictives ou non.

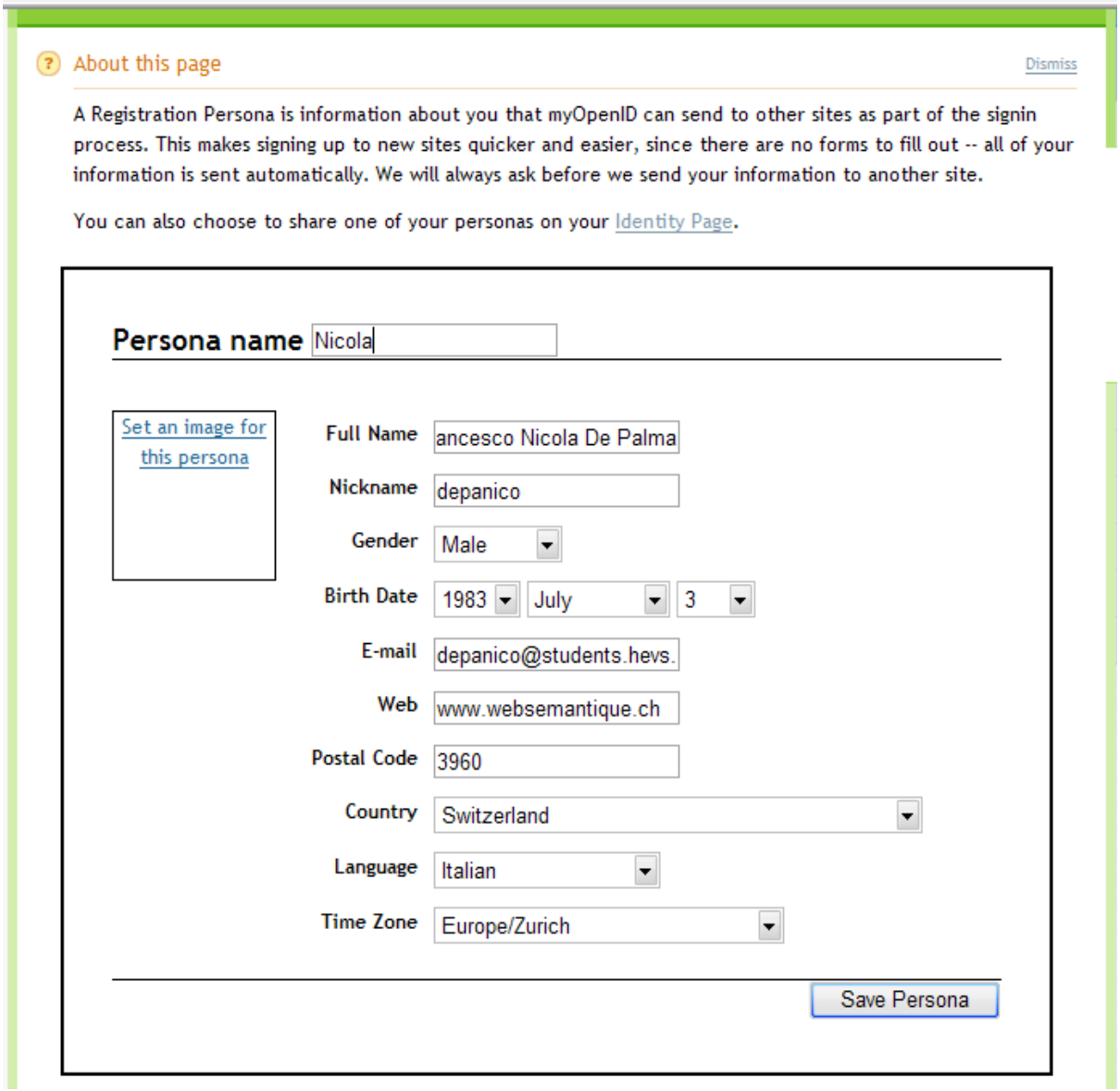


Figure 25 : OpenID création d'identité

Etape 6 : Première utilisation d'OpenID inscription sur un autre site

Après une inscription, il faut bien passer à l'étape d'utilisation d'OpenID dans les autres sites. Dans cette étape, l'utilisateur peut se connecter sur un autre site qui accepte l'authentification par OpenID et utiliser l'authentification par OpenID. Voici un site pris par hasard pour exemple : www.ziki.com qui permet d'obtenir ses propres pages web. On aperçoit tout de suite que les sites donnent la possibilité de s'inscrire de deux manières différentes : avec OpenID ou avec ancienne méthode qui demandera pour la nième fois un nickname, une adresse email (qui pourrait être spammé si site pas de confiance), un password....

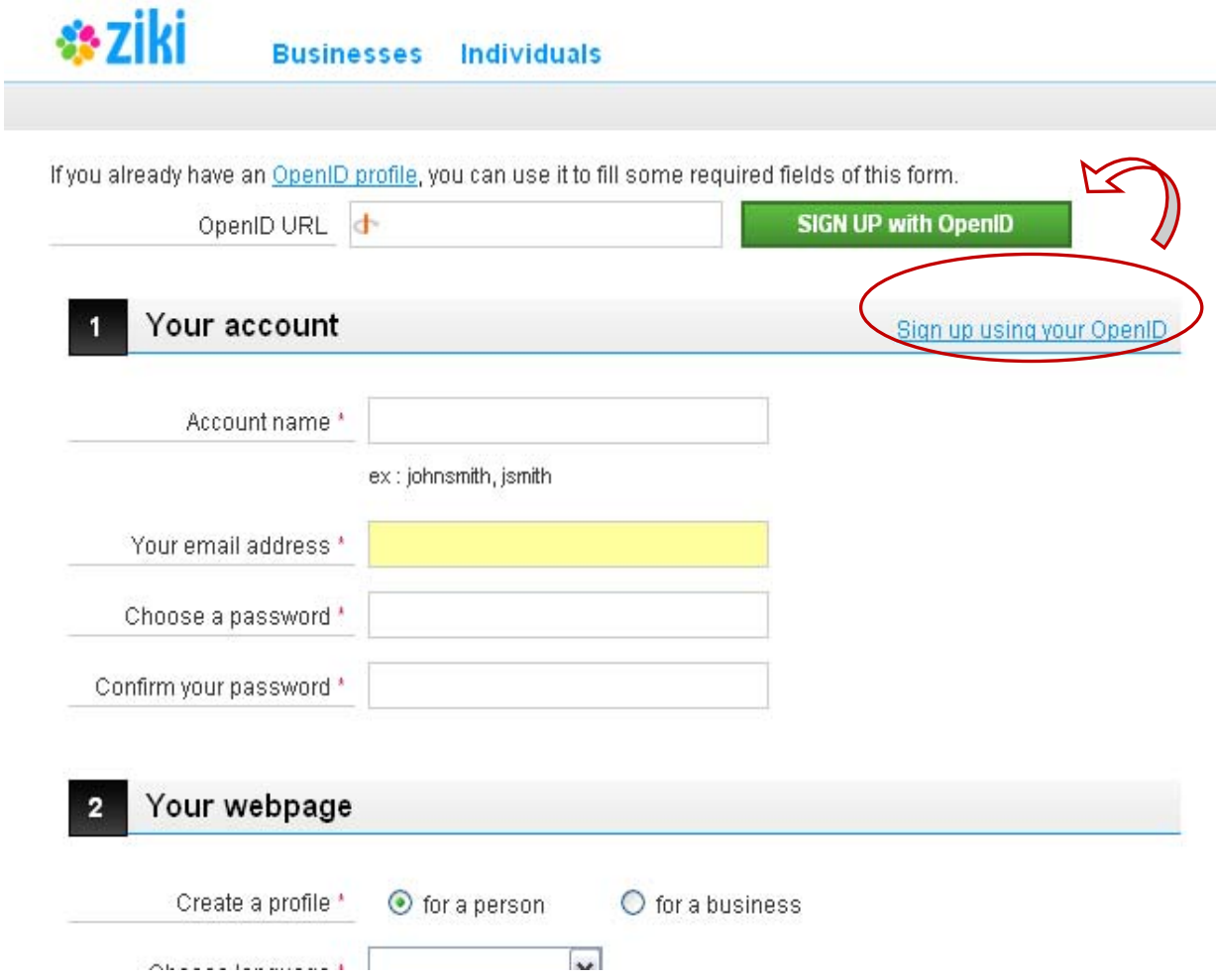


Figure 26 : Login OpenID de ziki.com

Etape 7 : Login avec OpenID

L'utilisateur est redirigé vers le fournisseur et doit saisir son mot de passe pour s'authentifier s'il n'est pas déjà logué.

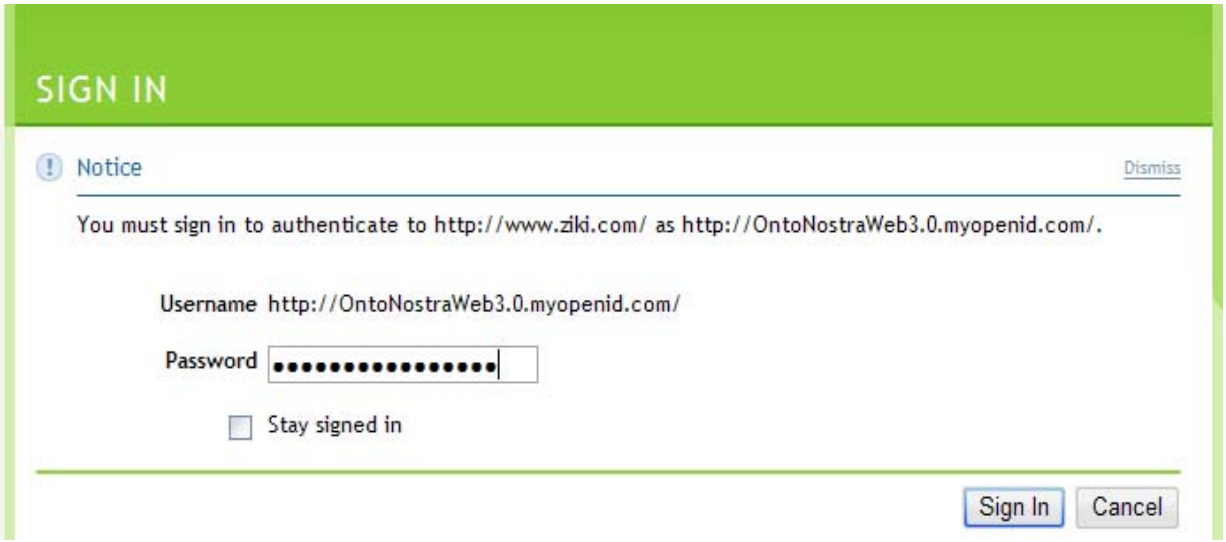


Figure 27 : Authentification d'OpenID

Etape 8 : Choix d'utilisation identité par un site

L'utilisateur est alors averti que Ziki souhaite avoir accès à certains attributs de son profil OpenID (nom, prénom, email…) Il aura trois possibilités de réponse possible :

- Accepter l'authentification dans ce site pour toujours
- Accepter uniquement pour cette fois
- Refuser

Dès lors s'il choisit un des deux premiers choix, donc s'il accepte, il pourra utiliser soit :

- Une des identités déjà crée, soit
- Créer une autre.

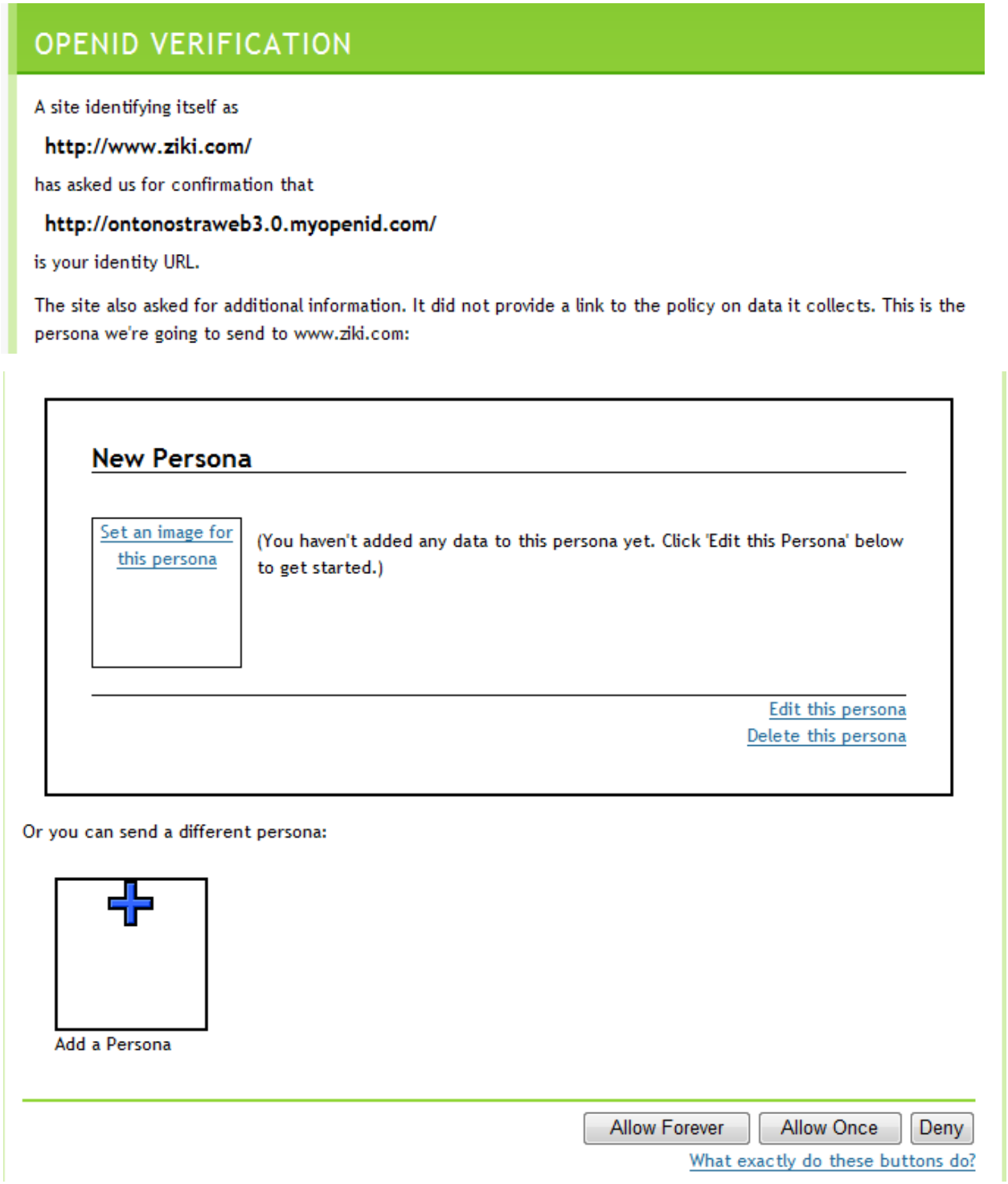


Figure 28 : Choix d'authentification à un site

Tant que la session de l'utilisateur est active, il pourra être reconnu automatiquement sur les autres sites en utilisant OpenID grâce au mécanisme d'authentification unique. Grâce à la page de configuration, l'utilisateur pourra à tout moment changer d'idée.

6.1.2. CONCLUSIONS

OpenID est un nouvel aspect du Web 3.0 qui tente de simplifier au maximum la gestion des profils d'authentification Web (noms d'utilisateur et mots de passe). L'URI est utilisé pour identifier les utilisateurs. Il semble très judicieux de l'appliquer dans notre cadre afin de représenter les URI des personnes, par exemple dans les documents FOAF. Une visite sur http://openiddirectory.com/ donne une idée du nombre de sites, où l'on peut utiliser OpenID. Au moment de la rédaction de ces lignes, il faut déjà compter sur 587 sites pour 42 catégories diverses de sites. Pour bien faire comprendre qu'OpenID n'est pas une idée restreinte à une niche de personnes. Des grandes entreprises comme Yahoo acceptent maintenant OpenID et il se dit même que Microsoft songe à le mettre en place.

Points forts

- Possibilité d'utiliser l'identité créée dans d'autres sites donc de les recycler
- Ne plus devoir faire de contrôles pour l'URI d'une personne : l'unicité serait protégée.
- Gratuit et facile d'utilisation.
- Grande communauté de développeur
- Variété de sources et de langages (Java, PHP, C#, Perl...) offerts pour mettre en place son propre serveur et/ou son propre clients OpenID
- Choix total donnée aux utilisateurs de mettre quelles informations doivent être mis à disposition du publique. Aucune information obligatoire.

Points faibles

- Il existe actuellement un problème intrinsèque de sécurité avec OpenID. En effet le site sur lequel l'utilisateur se logue avec OpenID aura toute la possibilité pour faire ce qu'il veut sur le serveur OpenID ainsi que sur tous les autres sites sur lesquels il est connecté avec OpenID. Cependant, l'utilisateur pourra saisir quel site s'est montré déloyal en abusant de ses données personnelles.

6.2 SIMILE ET SES OUTILS

Pour faciliter et sécuriser l'utilisation d'OpenID, je conseille un petit utilitaire qui se trouve dans les nombreux plugins proposés par le monde Firefox. Il se nomme **Appalechien** Firefox et facilite l'utilisation d'OpenID. C'est un gestionnaire d'identité qui permet d'avoir plusieurs OpenID qui ont plusieurs identités et ajoutent une fonction de sécurité par dessus.

Cet outil fait parti d'un grand projet **SIMILE** (**S**emantic **I**nteroperability of **M**etadata and **I**nformation in un **L**ike **E**nvironnements). Ce projet web du MIT crée et développe de nombreux utilitaires Open Source expressément dédiés au Web Sémantique et à sa technologie. Voici une petite liste avec leur principal fonctionnement :

- **RDFIZERS** : est une collection d'outils qui convertissent les données et fichiers en RDF (JPEG ->RDF, Email -> RDF, JAVA-> RDF...)
- **Piggy Bank** : est une extension du browser Mozilla Firefox. Piggy est un outil web le plus intéressant et utilisé du projet SIMILE. Piggy Bank (image à côté) extrait les données de vos sites préférés et les rassemble. Il peut même stocker les données localement pour donner la possibilité de les consulter par après et de les échanger. Tout cela au format dédié au Web sémantique RDF.

- **Semantik Bank** : est le serveur interne compagnon de Piggy Bank qui est utilisé pour le travail de stockage et l'échange d'informations
- **BABEL** : est un outil pour convertir des fichiers de web sémantique entre eux par exemple (RDF->N3, N3->RSS...)
- **Gadget** : est un inspecteur de fichier XML
- ...

Et plein d'autres d'utilitaires très intéressants toujours dans le monde du Web Sémantique, pour de plus amples informations référez – vous, svp, au site officiel en évolution continu du SIMILE http://simile.mit.edu.

7. WEB SERVICES

Voilà arrivé le chapitre sur l'unique partie qui a demandé un peu de programmation : Web Services. Il représente le seul moyen pour voir le fonctionnement de l'architecture, c'est l'intermédiaire entre OntoMea et OntoNostra. En guise d'introduction, il paraît essentiel à mes yeux de définir ce qu'est un Web service. Il s'agit de composants métiers qui s'exécutent à travers l'internet ou l'intranet, auto-suffisants et auto-descriptifs (dans le sens où toute la description se trouve avec le travail proposé). L'interopérabilité entre différents agents est le travail principal des Web Services.

Les Web Services sont souvent associés à l'échange de messages SOAP (expliqué brièvement au chapitre sur **l'architecture**) via HTTP. En revanche, une approche alternative des services Web qui devient de plus en plus populaire : REST. En effet, REST (comme le style d'architecture cité au chapitre 4) reprend les mêmes concepts de ce style d'architecture. Il permet l'envoi de messages sans enveloppe SOAP en étant ainsi moins lourd et dans un encodage libre (XML, binaire, simple texte).

REST est actuellement très utilisé par les sites communautaires appelés également réseaux sociaux. Ces derniers peuvent ainsi proposer à leurs clients une API facile à utiliser. Une **API** (Application Programming Interface) permet de définir la communication entre différents composants informatiques. Des sites typiques du **Web 2.0** comme *Flickr, Facebook, Last.fm ou Amazon* proposent des API Rest pour éviter à leurs clients de devoir passer par la case SOAP plus difficile à mettre en place. Nous avons repris l'idée des Web Services en REST dans OntoNostra. Les fonctions proposées par les Web Services actuellement développées pour OntoNostra sont :

- Ramener tous les graphs d'assertion sur le serveur central quand les OntoMea le demandent
- Importer et exporter les données (Images, musique, document…) et leur attribuer un URI correcte.
- Lister les fichiers qu'un utilisateur à mis sur OntoNostra
- Importer et exporter des triplets par l'intermédiaire des fichiers du type RDF.
- Recherche de l'information dans le serveur central OntoNostra, les autres OntoMea et d'autres ressources disponibles dans le Web comme Dbpedia ou GeoNames qui retournent des triplets.
- Créer et envoyer les certificats aux machines clientes

Ces fonctions se retrouvent dans quatre web services créés avec Axis : Nom des services proposé dans ce projet avec le nom des méthodes et les variables requises:

- **webservice_Certificat**
  - **Public byte[] donneCertificat(String NameUser, String OntomeaName, String eMail, String Password)** Renvoie un certificat et l'ajoute dans le Keystore du serveur en ayant le Nom de l'utilisateur, le nom d'OntomeaName (impossible d'avoir à double), l'email, le mot de passe. Ce certificat est très important car sans l'identification des OntoMeas aucune des autres fonctions n'est possible.
- **webservice_Envoi**
  - **Public String listeFichier (String OntoMea**) renvoie un string avec une liste des fichiers stockés dans OntoNostra gràce à l'identification d'OntoMea.
  - **Public byte [] renvoyerFile (String OntoMea, String FileName)** Après avoir choisi le nom du fichier le web service renvoie aux clients les fichiers stockés dans OntoNostra.
- **webservice_Recevoir**
  - **Public void recevoirMetadonnee (byte [] f, String OntoMea)** Reçoit les métadonnées (fichier .rdf ou. Owl) qu'un client désire envoyer et le charge dans la base de connaissance d'OntoNostra.
  - **Public void recevoirFichier (byte [] f, String OntoMea, String Filename)** Reçoit les fichiers qu'un client désire envoyer à OntoNostra et les classe dans un répertoire se nommant comme l'identificateur d'OntoMea et le nomme selon le désire de l'utilisateur.
- **webservice_Recherche**
  - **Public String envoiUri (String Uri)** grâce à un Uri renvoie tous les URI des fichiers Rdf contenant cette ressource dans ses variables

Une description plus approfondie relative au travail et l'utilisation des Web Service est présente dans le chapitre des **scénarios**. Pour la partie technique, le travail en effet a utilisé Tomcat comme serveur Web avec Axis conteneur de Web Service de plus amples informations à 12.3. Pour relier les différents OntoMea, pour l'instant, en utilisant uniquement avec le serveur OntoNostra.

8. OUTILS UTILISES

Durant la recherche de sources d'exemples et de logiciels utiles au projet, le travail m'a apporté à découvrir plusieurs outils très intéressant, utilisés dans le monde Open Source. Vous pouvez avoir un aperçu des nombreux utilisé dans une figure à la fin du document *(cf. Figure 42 : Outils utilisés)*. Voici une brève présentation des principaux outils :

8.1 TOMCAT APACHE

Apache et Tomcat est le résultat d'une combinaison entre deux programmes. **Apache** est le serveur Web le plus utilisé dans le monde plus de 70% des sites construits avec lui. Apache est un logiciel permettant à des clients d'accéder aux pages Web installé sur un ordinateur distant. **Tomcat** est, en revanche, un serveur d'application dédié à Java. Il possède une couche dédiée à la liaison avec le serveur Web Apache.

Pour l'élaboration de mon travail, j'ai utilisé le duo Tomcat Apache afin d'ajouter une couche supérieure d'accès entre les OntoMea et OntoNostra. Le rôle principal du duo est l'**échange d'informations** entre eux via des Web Services créé et implémenté avec Axis (cf. 8.2) intégrés à Apache/Tomcat. Un deuxième rôle de ce duo est la **sécurité** : la partie https et l'encryptage des données sont ajoutés. Vous trouverez l'installation et configuration de Tomcat à la section 12.3.2.

8.2 AXIS 2.0

Axis, dont la version actuelle est 2.0, est un Framework utilisé pour créer, déployer et consommer un Web Service et écrit en Java. Axis est, comme cité précédemment, intégré à Apache Tomcat.

Le projet a utilisé Axis comme conteneur et aide au développement des Web Services créés pour le projet toujours en langage Java. Il s'agit, par conséquence, de la partie de requêtes et de réponses envoyées, d'échange de données entre OntoMea et OntoNostra. Vous trouverez l'installation et la configuration d'Axis à la section.12.3.3.

Wsdl2java et Java2Wsdl

Axis2 a créé un fichier bat qui se nomme **WSDL2JAVA**. Ce fichier bat, comme son nom l'indique transforme un fichier WSDL en classe Java utilisable pour l'application cliente. Chaque Web Service possède un WSDL acronyme de Web

Service Description Language qui décrit la façon dont le service travail. Il existe également le fichier bat qui exécute le travail inverse **JAVA2WSDL**.

A noter qu'un plugin à ajouter au programme Eclipse (ide complet dédié à Java) reprend ces mêmes fonctionnalités sans passer néanmoins par ligne de commande Par ce fait, il est plus facile à utiliser. Visitez la page html d'axis http://ws.apache.org/axis2/tools/1_0/eclipse/wsdl2java-plugin.html.avec installation et aide à l'utilisation

8.3 SVN (SUBVERSION)

Le second outil est subversion **SVN**. Il s'agit d'un système de gestion de versions qui agit sur une arborescence de fichiers afin de conserver les versions des fichiers. Ce système est très utile pour le travail en équipe car il évite des éventuelles confusions. Il conserve en effet une trace de toutes les modifications. SVN est également un moyen très utile pour avoir toujours les dernières versions des sources,

J'ai utilisé SVN dans mon travail afin d'explorer et de découvrir les différentes sources mises à disposition par les librairies, entre autres, d'OpenID, SIMILE ou encore les Web Services et pour être à jour avec les sources proposées. Pour de plus amples informations veuillez vous référer au site officiel de SVN http://subversion.tigris.org/

8.4 ANT

Le troisième outil est un projet apache **ANT**, acronyme d'Another Neat Tool, qui signifie « un autre chouette outil » donc pas fourmis en Anglais. Celui-ci fonctionne sur tous les types d'OS. ANT est un logiciel qui automatise toutes les tâches répétitives dédiées au langage Java, qui exécute le travail d'une carte géographique servant à indiquer où aller pour trouver ces informations. Ainsi un fichier XML, **build.xml,** est créé dans le projet pour qu'Ant puisse fonctionner.

En outre, il donne la possibilité de compiler les projets avec recherche des librairies, la génération de documents (Javadoc), la création de jar, war (des jars pour le web) ou de fichier aar. Les fichiers .aar sont des jars dédiés aux Web Service afin d'être uploadé dans axis2. De nombreux programme intègrent Ant dans leurs propriétés.

J'ai chois d'utiliser ANT car il est fréquemment utilisé lors de projet similaires. En effet les exemples donnés, entre autre, par la librairie AXIS ou Tomcat sont tous construit au moyen d'ANT. Pour de plus amples informations veuillez vous référer au site officiel d'ANT http://ant.apache.org/. Installation et configuration d'ANT à la section 12.3.1.

9. STOCKAGE

Une problématique supplémentaire a été le mode de stockage point affronté durant le travail :

- Implémentation d'un système de stockage et de mise à jour en faisant attention à la faisabilité technologique et à la performance.
- Possibilité d'ajouter des données provenant du Web.
- Gérer la mise à jour des données en tenant compte des contraintes du web sémantique
- Tests de montées en charge avec des banques de données conséquentes (GeoNames, Dbpedia,…)

Le travail aurait pu effectuer le même choix fait pour la partie local, c'est à dire la base de données relationnelle « **HSQLDB** » avec Jena. HSQLDB est un système de bases de données embarquées et très légères écrit en Java se trouvant dans l'OntoMea local. Ou encore une autre possibilité est de me tourner vers d'autres modèles de stockage dont voici les trois types principaux :

- Le **mapping** (transformation) à la volée d'une modélisation d'une base de données relationnelle normale et du modèle de graphe RDF. Ce modèle est celui des outils **D2R server** ou SquirellRDF, par exemple.
- Le **stockage** dans une base de données relationnelles dont la modélisation permet le stockage direct sous forme de triples, par exemple : 3store, RAP ou ARC reposant sur MySQL, Virtuoso, Sesame ou Semantic Web/RDF Library for C#/.NET, **JenaSDB**

- Les **entrepôts** qui stockent de façon native les triples RDF, par exemple : AllegroGraph, BigOWLIM qui peut être une des solutions de stockage et d'inférence pour Sesame, Oracle 11g, Mulgara. Les deux derniers n'offrent pour l'instant qu'un support limité de SPARQL.

Dans la liste ci présente, j'ai choisi d'approfondir deux des plus connus mis en évidence et plus utilisés dans le monde du Web Sémantique, **JenaSDB** et **D2R server**. J'ai choisi de mettre côté le modèle de stockage « **entrepôts** » car il est moins performant et plus gourmand en espace disque. La différence majeure entre le mode **Mapping** et **Stockage** est la modélisation de la base de données relationnelles et l'effort à procurer pour réaliser des RDF. Le « Mapping » utilise une modélisation non-nécessairement dédiée aux triplets qui forment des fichiers RDF contrairement au mode « Stockage » qui lui demande une modélisation spécifique à RDF.

9.1 D2R SERVER

D2R server donne la possibilité aux browsers (internet, Firefox…) et au browser RDF de naviguer dans les contenus d'une base de données relationnelles (qui ne reprennent pas le fonctionnement d'un RDF) publiés dans le web. Il permet aux applications d'interroger SGBD (Système de Gestion Base de Donnée) utilisant SPARQL à travers le protocole SPARQL. Pour se faire, il s'agit d'avoir un mapping (transformation) entre la source relationnelle et la représentation ontologique de la base. **D2RQ**, utilisé par D2R-Server, est un langage de mapping adapté à cette fonction qui permet de traiter la source relationnelle non-RDF comme un named Graph RDF virtuel. Les named Graph sont caractérisés par des assertions qui sont des triplets.

L'élément principal de D2RQ est le "d2rq:ClassMap", lequel représente le mapping d'un ensemble d'entités décrites à l'intérieur du SGBD (Système Gestion de Base de Donnée) à une classe ou groupe d'identité similaire de ressource. Le document RDF généré à travers le langage D2RQ, qui en définitif décrit l'ontologie de la base de donnée, est créé en utilisant un instrument intégré à l'intérieur de D2R-Server qui génère un mapping à travers les tables du SGBD. Grâce à ce système, les applications clientes du Web Sémantique peuvent récupérer la représentation RDF et HTML des ressources et peuvent également interroger la base de donnée non-RDF à travers SPARQL. L'architecture de D2R est présentée plus particulièrement à travers l'image ci-dessous.

La représentation des données générées par D2R sont fortement interconnectée au niveau de RDF et HTML pour rendre le plus simple possible et interchangeable la navigation du contenu de la base (*cf. Figure 29 : Architecture de D2R server [19]*). En effet, l'instrument fourni une véritable et propre fonction d'exploration des données. L'utilisateur n'est donc plus contraint d'écrire une requête pour rechercher les informations, puisqu'en utilisant la fonction d'exploration les query SPARQL sont écrites au fur et à mesure automatiquement.

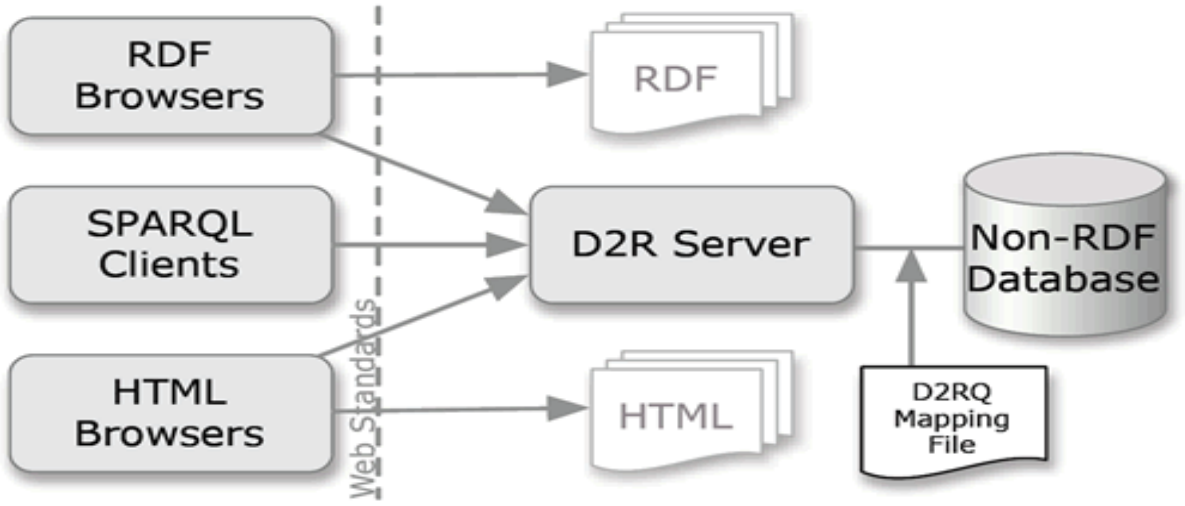


Figure 29 : Architecture de D2R server [19]

9.2 JENA SDB

Avant d'articuler la partie SDB de Jena, il me paraît important de traiter du projet Open Source **Jena** http://jena.sourceforge.net/. Le Framework web sémantique écrit en Java est destiné au développement d'applications dédiées au Web 3.0. Jena offre de nombreuses librairies pour la gestion des standards (expliqués précédemment) **RDF**, **RDFS**, **OWL** et **SPARQL** et inclut un moteur d'inférences basées sur les règles de déductions.

Développé par le «HP Labs Semantic Web Programme ». Jena inclut un mini serveur, **Joseki,** qui supporte le protocole SPARQL. **ARQ** est, quant à lui un moteur de recherche qui supporte SPARQL et qui retourne les résultats. A ce sujet, vous avez à disposition une démonstration http://www.sparql.org/query.html. Et encore, **Eyeball**, qui contrôle les modèles RDF et OWL dans le but de résoudre les problèmes de modélisation.

SDB est destiné au stockage d'informations RDF. Il comporte un interpréteur de requêtes SPARQL, basé sur le moteur **ARQ** et qui permet de transformer ces requêtes en SQL standard. SDB store peut être utilisé par ligne de commandes ou par l'API de Jena. Le principal avantage, par rapport au stockage de données de Jena, est un gros gain en performance grâce à des algorithmes gérant les triplets de manière optimisée. Voici à présent une présentation des deux layouts (deux types de modélisation) offerte par Jena SDB layout1 et layout2. Layout1 est le schéma le moins performant mais compatible avec toutes les bases créées avec Jena. Il est également plus facile à utiliser notamment grâce à sa flexibilité. Comme vous pouvez remarquer il est composé de deux tables :

- La table **Triples** qui reprend le fonctionnement d'un triplet {**s**=sujet, **p**=prédicat, **o**=objet} et dont les 3 colonnes ont un rôle de clé primaire.
- La table **Préfixes** qui stocke les URI et utilise préfixes comme clé primaire

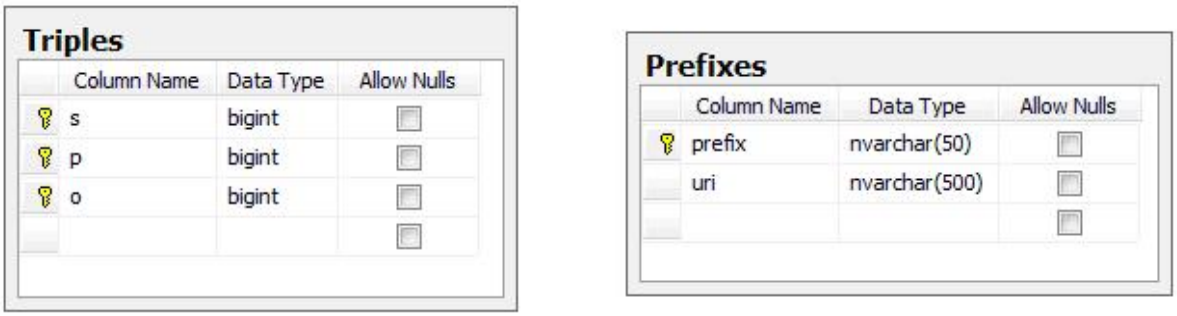


Figure 30 : Diagramme du layout1

Le layout2 est propre à SDB. Il est utilisé pour accroître les performances notamment dans la phase d'écriture, allant jusqu'à dix fois plus vite. Cependant, il ne peut pas être utilisé avec des données déjà existantes. En plus des tables du layout1 **Triples** et **Prefixes**, il est composé de deux tables supplémentaires :

- • La table **Quads** qui correspond à la table Triples du layout1 avec un champ supplémentaire pour la gestion des Named Graphs.
- • La table **Nodes** qui décrit en détails chaque élément du Named Graphs.

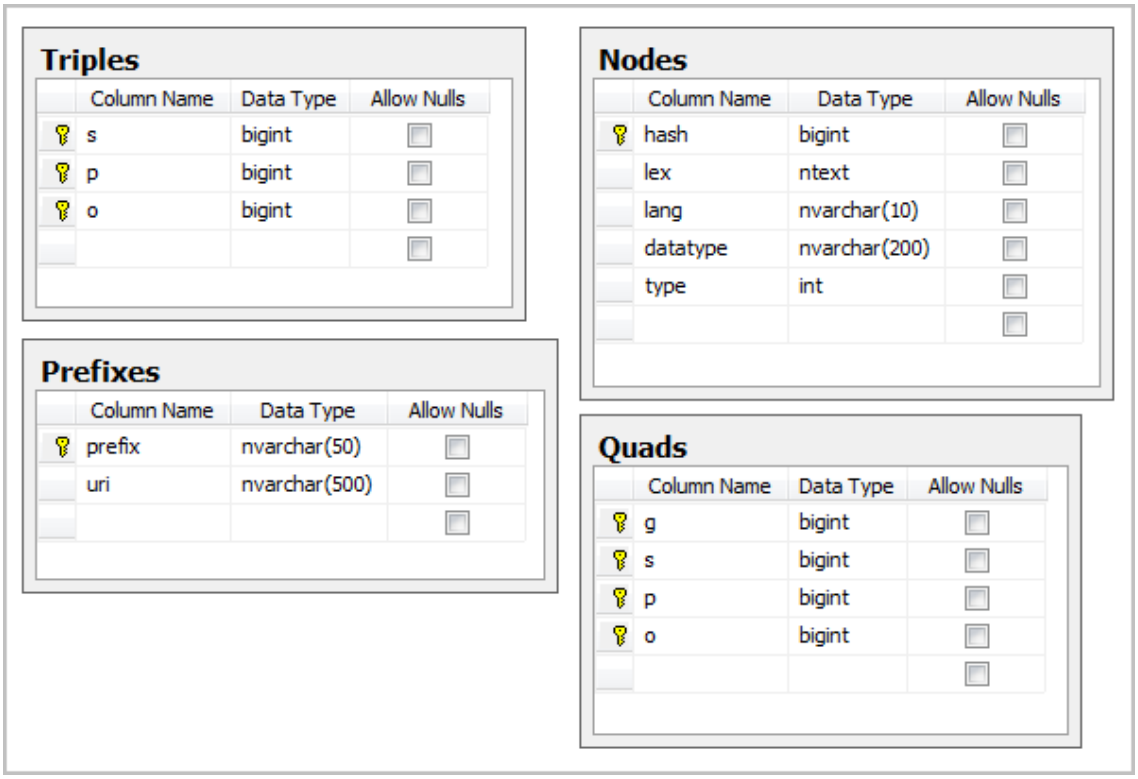


Figure 31 : Diagramme du layout2

Ci-dessous est présent un exemple de configuration de SDB pour notre serveur.

```
@prefix sdb:    <http://jena.hpl.hp.com/2007/sdb#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ja:     <http://jena.hpl.hp.com/2005/11/Assembler#> .

_:c rdf:type sdb:SDBConnection ;
    sdb:sdbType         "postgresql" ;
    sdb:sdbHost         "OntoNostra" ;
    sdb:sdbName         "OntoNostra" ;
    sdb:sdbUser          "user" ;
    sdb:sdbPassword     "password" ;
    sdb:driver          "org.postgresql.Driver" ;
[] rdf:type sdb:Store ;
    sdb:layout      "layout2/index" ;
    sdb:connection _:c ;.
```

Figure 32 : Configuration SDB

Les tables Triples et Quads contiennent ici les références sur la table Nodes. Le moteur travaille ainsi en grande partie avec des références ce qui implique un gain de performances important. La procédure d'insertion selon un test de Jena est d'environ 5000 Triplets par seconde avec un fichier RDF de Dbpedia qui avait 25 millions de Triplets. Pour de plus amples informations, veuillez vous retourner à la page de présentation du test : http://jena.hpl.hp.com/wiki/SDB/Loading_performance.

Dbpedia est une communauté ouverte qui crée une encyclopédie accessible aux machines. Elle reprend les données de Wikipedia et les transforme au format RDF/XML selon une ontologie bien définie. Dès lors, Dbpedia les stocke afin de pouvoir les interroger via un langage de requête **SPARQL,** et cela en plusieurs langues (Français, Anglais, Allemand…). Le site de Dbpedia se trouve à cette adresse : http://dbpedia.org/About.

SDB étant une couche par-dessus la base de données relationnelles, il est compatible avec un grand nombre de bases de données propriétaires et Open Source. Les développeurs garantissent le fonctionnement et donnent la configuration pour les bases de données suivantes

SGBD	Version accepté
Oracle 10g	Including OracleXE
Microsoft SQL Server 2005	Including MS SQL Express
DB2 9	Including DB2 9 Express
PostgreSQL	v8.2
MySQL	v5.0.22
Apache Derby	v10.2.2.0
HSQLDB	1.8.0

Tableau 6 : Liste de SGBD utilisé dans Jena SDB

Il est malgré tout possible de configurer SDB pour n'importe quelle autre base de données, toutefois sans garantie de compatibilité.

Jena étant un Framework dans lequel OntoMea est déjà intégré. Les serveurs OntoMea et OntoNostra possèdent déjà Jena SDB dans ses nombreux outils proposés. Encore, il n'y aurait pas besoin d'ajouter de librairies et des couches par-dessus. Il s'agit donc d'un point positif en faveur de Jena et son SDB. Cependant, le choix doit se faire aussi sur le point le plus important dans le monde actuel la performance. Par conséquence, vous trouverez dans le chapitre suivant intitulé « **Conclusion** », un léger comparatif entre Jena SDB, D2Rserver ainsi que d'autres serveurs nommés dans le paragraphe précédent.

9.3 CONCLUSION

Je parlais précédemment de performance, ainsi il m'a fallu chercher comment prouver que l'un ou l'autre était le meilleur pour notre projet. En effet, après de nombreuses recherches voici une page internet d'un étudiant allemand (http://cweiske.de/tagebuch/SPARQL%20Engines%20Benchmark%20Results.htm) où se trouve une compétition entre :

- **RAP's** old SparqlEngine
- **RAP's** new SparqlEngineDb
- **ARC**, php implémentation
- **Jena SDB**
- **Redland**, C implementation. (1.0.6)
- **Virtuoso** Open Source Edition 5.0.1

Cette recherche affirme que Jena SDB est le seul à être complet et le seul qui puisse exécuter tout types de requêtes SPARQL et donc le plus performant. Après avoir lu cet article, j'ai tout de même voulu tester aussi celui qui manquait **d2rServer**. A la fin des tests, les performances étaient presque les mêmes. J'ai opté en revanche pour Jena SDB pour des questions **d'intégration** avec les autres OntoMea.

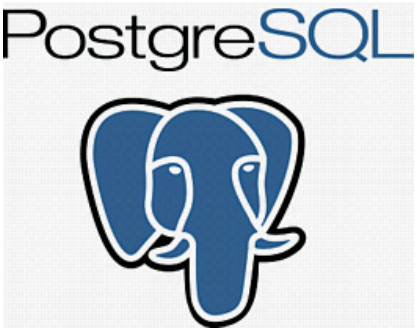
Après avoir choisi Jena SDB, il fallait choisir quelle base de donnée relationnelle fonctionne le mieux avec SDB au niveau des performances et des demandes de ressources. Après une recherche dans le forum des développeurs de Jena, voici le Link (http://tech.groups.yahoo.com/group/jena-dev/). J'ai ainsi pu tester les trois principaux SGBD : **SQL Server**, **MySQL** et **PostgreSQL**.

Selon le forum et les développeurs de Jena, les trois types de bases de données relationnelles travaillent tous très bien au contraire d'Oracle, qui a besoin de configurer pour améliorer ses performances. Un autre SGBD, HSQLDB qui, comme cité dans l'introduction, fonctionne déjà très bien en local est néanmoins un peu faible et léger pour un serveur central qui aurait à répondre à de nombreuses requêtes. Le choix s'est fait entre un produit open Source ou payant.

A parité de performance, j'ai choisi un produit Open Source, **PostgreSQL** que je présente brièvement au chapitre suivant. PostgreSQL est une base de données relationnelle qui est en pleine phase d'expansion et de développement et qui a un très bon écho dans le monde web. Il a été élu produit Open Source de l'année 2007 dans la catégorie de base de données.

9.3.1. POSTEGRESQL

Selon le site officiel dédié à **PostgreSQL en langue française** http://www.postgresqlfr.org/ voici une petite définition donnée à ce Système de Gestion de Base de Donnée :



*« **PostgreSQL**™ est un système de gestion de bases de données relationnelles objet (ORDBMS). POSTGRES est à l'origine de nombreux concepts qui ne seront rendus disponibles au sein de systèmes de gestion de bases de données commerciales que bien plus tard. **PostgreSQL**™ est un descendant Open Source du code original de Berkeley.*

*Il supporte une grande partie du standard SQL tout en offrant de nombreuses fonctionnalités modernes : requêtes complexes ; clés étrangères ; triggers ; vues ; intégrité des transactions ; contrôle des accès concurrents (MVCC ou multi version concurrency control). De plus, **PostgreSQL**™ est extensible par l'utilisateur de plusieurs façons. En ajoutant, par exemple :*

- *de nouveaux types de données ;*
- *de nouvelles fonctions ;*
- *de nouveaux opérateurs ;*
- *de nouvelles fonctions d'agrégat ;*
- *de nouvelles méthodes d'indexage ;*
- *de nouveaux langages de procédure.*

*Et grâce à sa licence libérale, **PostgreSQL**™ peut être utilisé, modifié et distribué librement, quelque soit le but visé, qu'il soit privé, commercial ou académique.* »

En conclusion, PostgreSql offre tout ce qu'un Système de Gestion de Base des Données Relationnelles moderne doit apporter au développement. Il n'appartient pas à une seule entreprise c'est un groupe de développeur à travers le monde qui fait évoluer ses fonctionnalités toujours Open Source. PostgreSQL s'intègre très facilement à OntoNostra grâce à Jena SDB. La grande force de Postgre envers les autres SGBD est sa programmabilité grâce à de nombreuses bibliothèques qui s'intègrent très bien au monde Java.

10. SECURITE

La sécurité est un des problèmes majeurs dans le monde informatique et Web. Par conséquence, pour ce travail de diplôme dédié à l'interaction entre serveurs et clients, il s'agit de mettre en place une sécurité de fer. Sans cela, la communication serait trop difficile aux utilisateurs. Il est nécessaire de fournir quatre garanties majeures :

1. Etre sûr de son interlocuteur. C'est **l'authentification** réciproque des correspondants ou des composants.
2. Etre sûr que les données transmises n'ont pas été modifiées accidentellement ou intentionnellement. C'est **l'intégrité** des données.
3. Eviter que les données soient lues par des systèmes ou des personnes non autorisées. C'est la **confidentialité**.
4. Eviter la contestation par l'émetteur de l'envoi des données. C'est la **signature**, appelée aussi non répudiation.

Pour commencer à donner réponse à ses quatre garanties, j'ai ajouté une couche supérieure « **protocole ssl** ». SSL abrégé de Secure Sockets Layers est un standard. Le protocole SSL se base sur une infrastructure à clé publique (PKI) pour la double fonction d'authentification des intervenants et permettre les échanges sécurisés entre eux. Ce mécanisme prévoit que chaque participant à l'échéance ait une paire de clé (privé et publique). La clé privée vient jalousement tenue secrète ; vice-versa, la clé publique correspondante peut être distribuée librement. Les messages cryptés avec la clé publique peuvent être décrits seulement avec la clé privée correspondante.

Pour obtenir et stocker une clé, il est indispensable d'envoyer des certificats. Comme analogie, un certificat ajoute le rôle qu'à une carte d'identité ou un passeport, il donne la possibilité d'authentifier et de laisser l'accès. J'ai utilisé un des standards de cryptographie le plus utilisé **X.509**. La cryptographie est utilisée pour cacher et interdire d'accéder aux informations à des personnes malveillantes. Plus précisément, un certificat est un document qui permet d'attester qu'une clé publique vous appartient bien. Pour cela, il renferme plusieurs informations : votre clé publique, et des renseignements servant à vous identifier (votre nom, votre société, votre e-mail, la date de validité du certificat, ou d'autres champs supplémentaires, comme pour notre exemple le nom attribuée à l'OntoMea local). Pour créer les certificats, j'ai utilisé une bibliothèque de cryptographie java et Open Source : **Buoncy Castle** [44].

Dans l'image (cf. Figure 33), vous voyez un exemple plus concret de fonctionnement d'échange de clés et par conséquence de réseau sécurisé. L'échange de clé (privé/publique) rendra donc impossible à une personne malveillante (hacker) d'accéder aux informations échangés entre Alice et Bob à travers le réseau. La clé publique de Bob va être utilisée par Alice pour crypter son message ici « Alice ». La clé privé de Bob va décrypter se message dans le but que seul l'utilisateur ayant cette clé puisse lire le message.

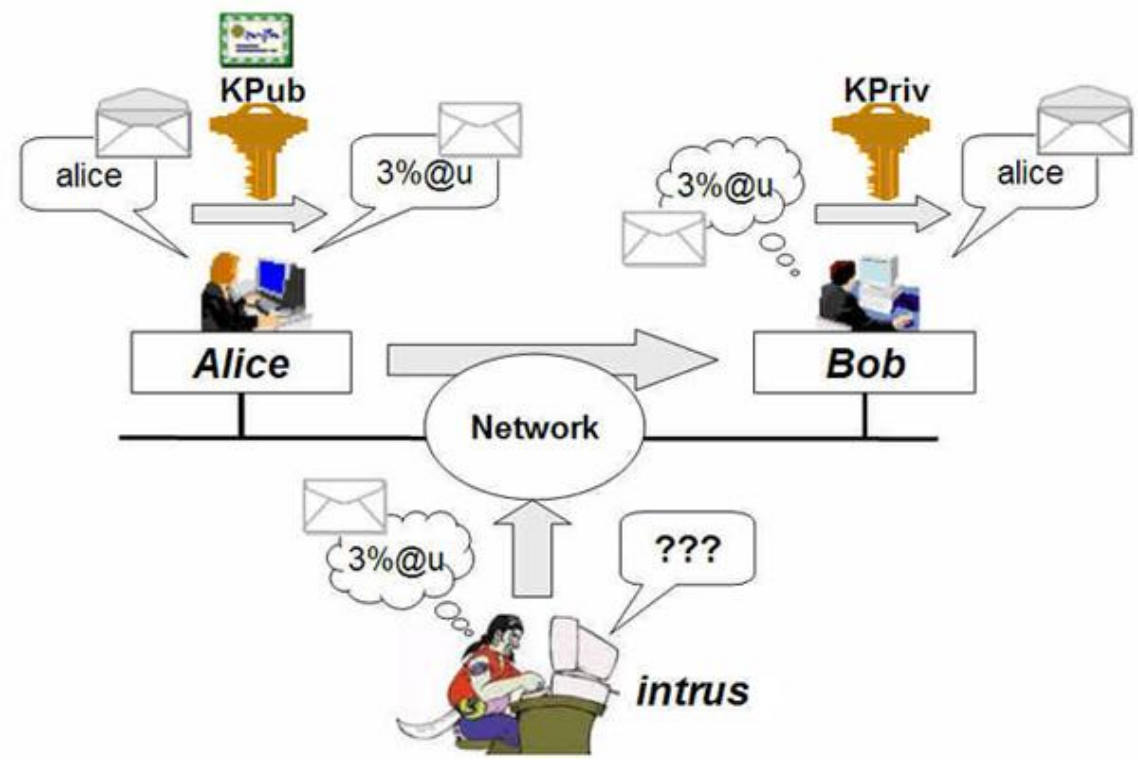


Figure 33 : Exemple réseau sécurisé [38]

Ces informations sont certifiées être justes par une autorité de certification appelé dans le web Certification Authority ou CA. L'autorité est censée avoir vérifié ces informations avant d'avoir validé votre certificat. La CA joue le rôle de l'agent des douanes qui vérifie votre carte d'identité avant de mettre le coup de tampon qui validera votre passeport. La CA hache et signe donc le certificat à l'aide de sa propre clé privée. Il suffit, par conséquence, de connaître sa clé publique (largement distribuée et notamment intégrée aux navigateurs Web) pour vérifier la validité d'un certificat généré par celle-ci. A la page suivante, se trouve une image qui explique le fonctionnement de cet échange.

Comme CA, j'utilise un outil intégré à java, le **Keystore** qui fonctionne comme une mini base de donnée dédié au certificat. Le Keystore qui est généralement un fichier avec comme extension .ks ou .jks, protège les clés privées avec un mot de passe. Le Keystore est créé grâce à **Keytool** un petit programme se trouvant dans le dossier bin de java et dans l'utilisation se fait à travers des lignes de commandes. Referez-vous au chapitre sur l'installation et configuration 12.2 pour avoir un exemple d'utilisation.

En ce qui concerne Rest, voyons ce qu'implique la sécurité. Grâce à l'idée d'URI et d'authentification, il suffit de vérifier que chaque utilisateur ou son agent (le serveur OntoMea) soit bien autorisé à accéder à l'URI spécifié et selon la réponse, autorisé ou non l'accès. **Par exemple**, l'utilisateur s'appelle Jean et s'authentifie comme tel avec son certificat, il ne pourra, par conséquence, pas accéder au page personnel de Nicola : http://www.semantique.ch/pageperso/nicola mais au contraire il aura le droit d'accéder à sa propre page grâce à l'authentification et l'URI : http://www.semantique.ch/pageperso/jean.

HTTPS qui signifie HyperText Transfer Protocol Secure est l'encapsulation du protocole HTTP au travers du protocole SSL. Il se voit attribuer par défaut le port 443 alors qu'HTTP le port 80. Le seul inconvénient réel est l'obligation de chiffrer l'intégralité des données de la page Web. HTTPS permet d'authentifier le serveur et de chiffrer les données transmises entre le serveur et un agent utilisateur. Ainsi, le client sera authentifié, dans notre architecture, avec un certificat X.509, dont voici une image d'un certificat type (cf. Figure 34 : Certificat).

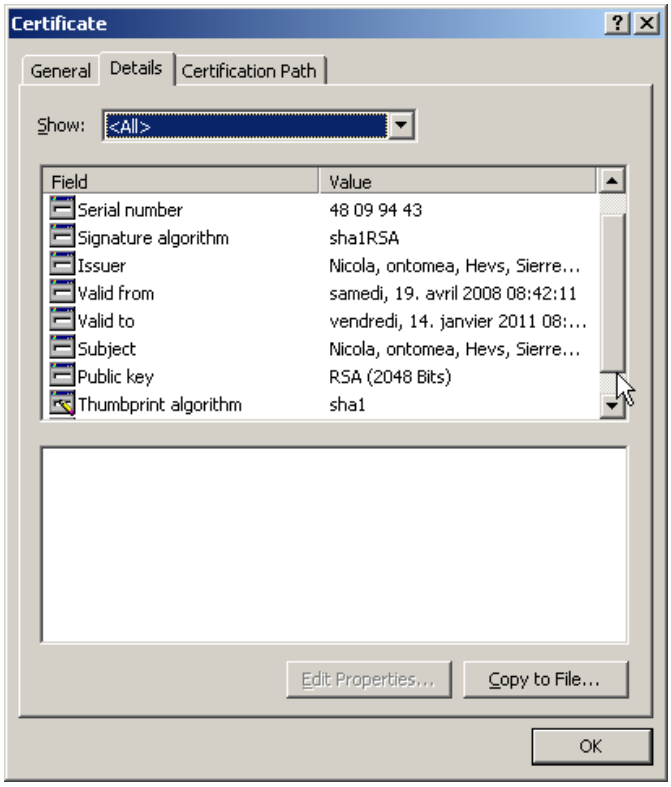


Figure 34 : Certificat

Pour conclure, ceci permet donc de répondre positivement aux quatre garanties citées précédemment (**authentification, intégrité, confidentialité, sécurité**). Le **certificat** permet de traiter de manière fiable le point 1 : l'authentification des correspondants. Le point 2 et le point 3 sont obtenus par le chiffrement **HTTPS**. Le point 4 est obtenu par l'utilisation de **certificats** par les utilisateurs. Ainsi, un très haut niveau de sécurité est facilement mis en place sur une architecture REST sans complexité supplémentaire.

11. SCENARIOS

Les scénarios sont des petits descriptifs du fonctionnement d'OntoNostra. L'idée est qu'un utilisateur local rapatrie toutes les informations nécessaires à une opération précise (un voyage, une recherche etc..), et que les déductions soient faites en local. J'ai choisi de présenter deux types de scénarios qui expliquent aussi le fonctionnement de l'architecture mise en place.

- Ajout d'informations
- Recherche d'informations

11.1 AJOUT D'INFORMATIONS

Une personne décrit ses données sur son OntoMea local. En indiquant par exemple qui se trouve sur une photo, où et quand a été prise celle-ci, etc.

Les fichiers qu'elle annote (images, fichier etc.) reçoivent un URI basé sur le **checksum**. Le checksum est généré à partir du contenu d'un fichier et non sur son emplacement. Ainsi, deux personnes qui annoteront la même image prise en voyage ensemble pourront décrire la même ressource.

La personne a le choix entre garder ces informations en **privé**, ou alors les partager avec un groupe d'utilisateurs à cercle fermé, voir les rendre **publiques** à tout le monde. Dans les deux derniers cas, les informations vont donc être envoyées au serveur OntoNostra.

Dans le cas des données privées, il est également possible que les informations soient envoyées sur le serveur. En revanche, elles ne seront pas accessibles aux autres utilisateurs, afin de pouvoir les récupérer sur un autre OntoMea de la même personne (passer des informations du bureau à la maison).

Le ***scénario***, User1 entre dans son OntoMea les informations relatives à son voyage en Australie:

- Des **triplets** décrivant le voyage, les étapes, ses activités, les médias ramenés, etc.
- Les **médias** eux-mêmes (des fichiers photos, sons, vidéos, etc.)

L'User1 indique que certaines des informations sur son voyage en Australie sont :

- **Publiques** : à dispositions du monde entier
- **Semi-privées** : disponibles uniquement pour un cercle d'amis ou de famille ou encore un groupe d'utilisateurs
- **Privées** : pour lui seul

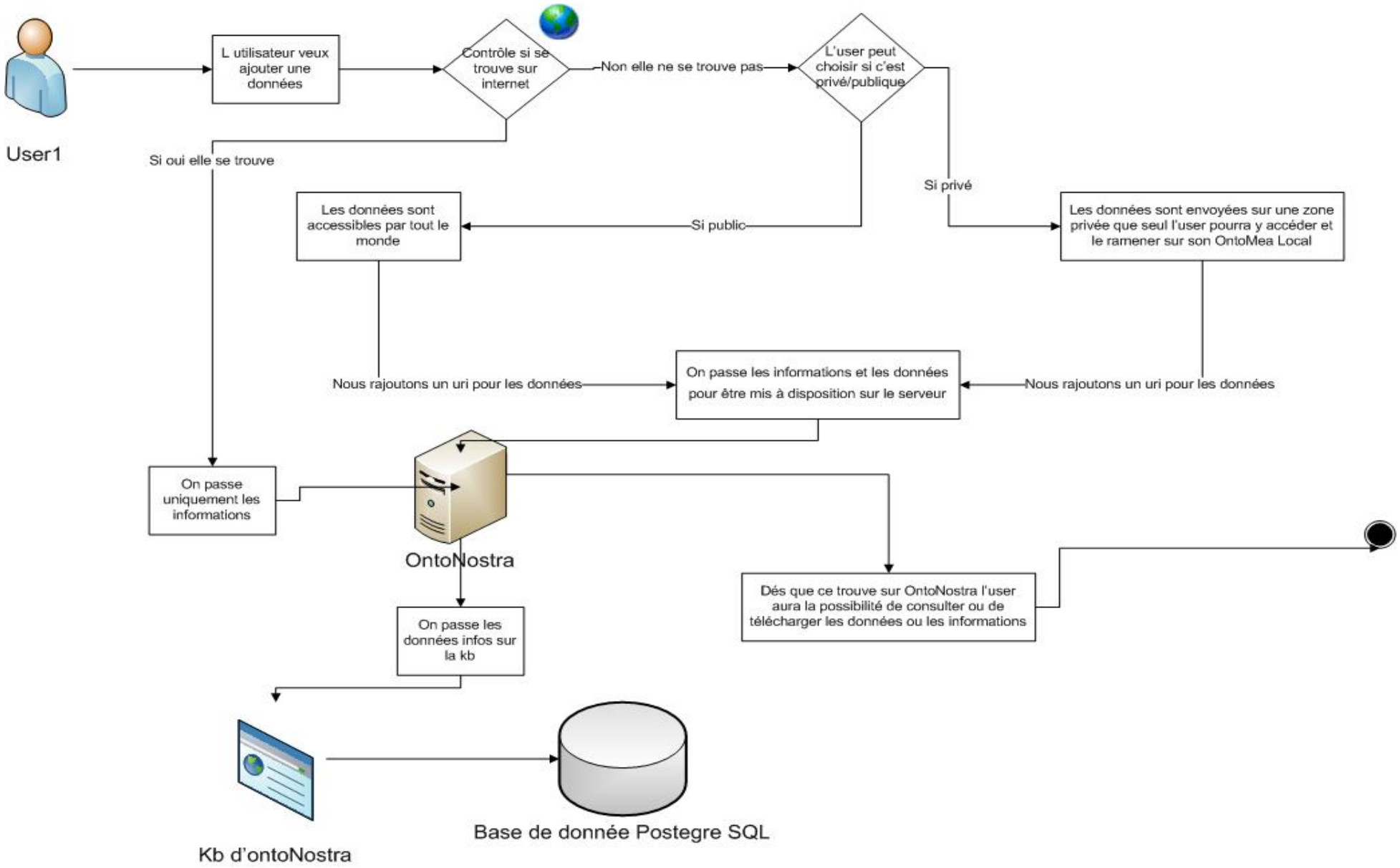
Les informations privées ne sont pas uniquement gardées dans l'OntoMea local. En effet, on peut également les transiter de deux façons/scénarios différents. Le premier peut s'effectuer entre un OntoMea d'un utilisateur, étant donné que ce dernier peut posséder deux PC dans deux lieux différents. Le transfert peut alors se faire à travers OntoNostra et ses Web Services.

Le deuxième scénario, un utilisateur est en voyage à l'étranger et veut mettre à jour des informations, Il va dans un cybercafé et à travers le Web et les Web Services, il enverra des infos **'privées ou publiques'** sur OntoNostra. Tout cela avec l'aide d'un site Web qui présentera les différents outils d'OntoNostra. Su retour de son voyage, il pourra, après une authentification dans le site Web, rapatrier les données en local dans son propre OntoMea.

Le serveur central va récupérer les informations d'User1 sur le voyage en Australie. Il charge les triplets dans son KB en leur attribuant un named-graph et un Uri (http://www.OntoMeaUser/namedgraphname) pour pouvoir gérer la mise à jour des informations, et charger les fichiers médias sur le serveur. L'URI est capital pour maintenir les informations à jour : la KB (knowlodge base, base de connaissances) est un graph composé d'un sous-ensemble de named-graph. On peut ensuite lors de la mise à jour décharger le contenu d'un named-graph (=contenu de la version précédente du fichier), puis charger le nouveau fichier.

En tout temps, User1 doit pouvoir indiquer au serveur les mises à jour qu'il a effectuées afin que celui-ci puisse en tenir compte. **OntoNostra** contient alors des informations relatives à l'Australie très complètes, provenant de tous les utilisateurs ayant alimentés le système.

Web 3.0 : Déploiement



11.2 RECHERCHE D'INFORMATIONS

Une personne désire consulter toutes les informations disponibles par rapport à une ressource (une photo, un pays), par exemple en vue d'organiser un voyage par exemple.

L'OntoMea local va demander au serveur OntoNostra de lui faire parvenir toutes les données RDF nécessaires. Une fois les données rapatriées, des déductions pourront être faites pour enrichir ces données de base.

Le même moteur se trouvant au cœur de l'OntoMea local et du serveur OntoNostra, les fonctionnalités sont très proches entre le scénario d'ajout d'informations et celui-ci. Comme il est indispensable de pouvoir identifier précisément la source des informations pour les mises à jour futures, ces échanges se font par web-services définis, et non pas par un SPARQL end-point (un mode de faire des requêtes SPARQL). En effet, il fallait définir des web-services spécialisés qui exportent toutes les données propres à une ressource et ainsi à une URI (toutes les données sur une photo, un pays): je prépare un voyage, donne-moi toutes les informations utiles pour un voyage dans telle région (et le web-service va renvoyer les informations sur le pays, les photos et commentaires d'autres voyageurs, etc.)

Le ***scénario***, l'User 2 recherche des informations sur l'Australie. L'utilisateur va envoyer des requêtes à OntoNostra. L'utilisateur va s'authentifier grâce à OpenID et le serveur, quant à lui va aussi s'identifier et authentifier grâce au certificat reçu précédemment.

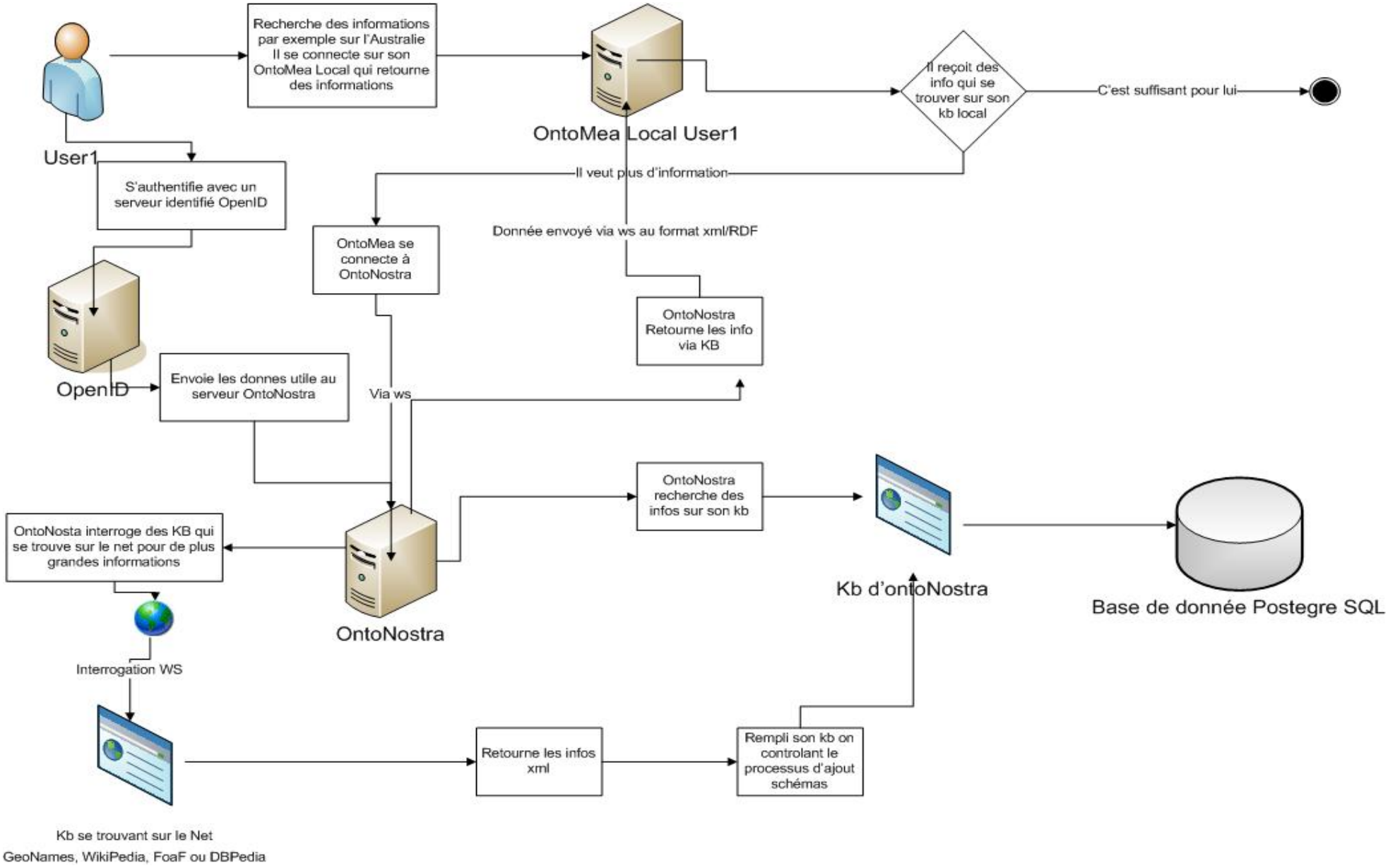
Les informations reçues seront rapatriées en local sous formes de fichiers RDF et de données multimédias selon le type de recherche.

Toutes les informations ne se trouvent pas sur OntoNostra. Par exemple il faudrait aller chercher certaines informations RDF sur un serveur tel **GeoNames** ou **Dbpedia** ou encore dans d'autres OntoMea locaux. OntoNostra servira d'intermédiaire des chercheurs d'informations. Ces informations seront simplement envoyées à l'OntoMea de User2 sans les garder en cache sur le serveur central.

Les données **d'un OntoMea** au sujet **d'une ressource** sont ainsi identifiées par **un named graph**. Pour gérer les updates, il est important qu'un named graph contienne toujours les mêmes données, et par conséquence, toutes les données d'un OntoMea relatives à une ressource. On transfert donc tous les triplets où la ressource est sujet ou objet.

Pour cette raison, l'échange de triplets entre les triples stores s'effectuera par fichier : pour envoyer les informations au sujet d'une photo, l'OntoMea local sauve tous les triplets dans un fichier RDF, puis indique au serveur central de venir chercher ces informations. Le serveur central utilise le chemin du fichier comme named graph, regarde si ce named graph n'est pas déjà chargé, auquel cas il décharge l'ancienne version, puis charge le nouveau.

Web 3.0 : Déploiement



12. INSTALLATION ET CONFIGURATION

Maintenant, après avoir bien expliqué les concepts, il faut essayer d'expliquer les configurations et les installations qu'a demandées ce projet. La configuration de ces différents produits a pris énormément de temps car la documentation était très dispersé et très difficile de compréhension. Le travail présente ici uniquement les points les plus importants affronté durant tout le projet. L'installation (avec la configuration) se fait en trois grandes parties qui représentent les chapitres traités précédemment. J'ai mis entre parenthèse les sections où se référer :

- Base de données PostgreSQL (Stockage)
- Création Keystore (Sécurité)
- Serveur et Client Web (Web Services et Architecture)
    - Configuration d'Ant (Outils Utilisés)
    - Installation et Configuration (Sécurité) Tomcat
    - Installation et Configuration (Sécurité + REST) Axis

A noter que Rest et l'identification n'ont besoin d'aucune installation particulière, étant donné qu'il s'agit de règles, de méthodes et de choix faits durant le projet. Il sera néanmoins nécessaire de configurer les outils. Pour ce qui est d'OpenID et de ses fonctionnalités, il est uniquement nécessaire d'implémenter un client OpenID dans le site http://www.websemantique.ch.

Toutes les sources, se trouvent dans le répertoire **Download** du CD et à disposition via le Web.

12.1 BASE DE DONNEES

L'installation va commencer par peut être ce qui est le moins ardus l'installation du SGBD PostgreSQL (cf. section 9.3.1). Voici expliqué en quelques brèves étapes les démarches à faire.

- Installer PostgreSQL, dans ce projet est inclus la version 8.2. Après l'avoir configuré selon les variables par défaut.
- Cliquer dans l'admin de PostgreSQL authentifiez-vous et créer un utilisateur, dans cette exemple : « test » et une base de données, nommé ici : « OntoMea ».

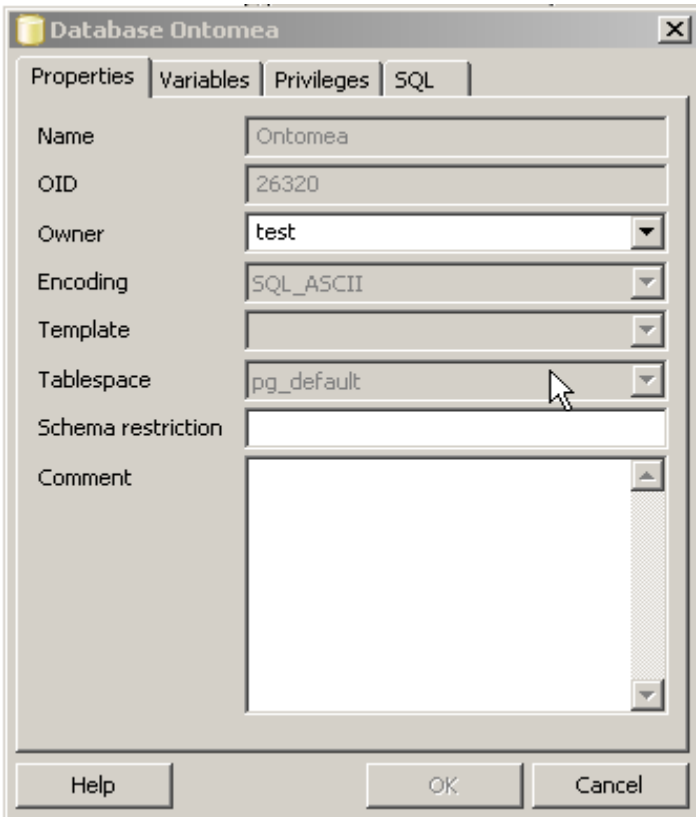


Figure 35 : Propriété base de données

Ne créer aucune table, ça sera Jena SDB fera le travail pour vous. Il ne restera à choisir quelle layout (quelle modélisation) avoir (cf. section 9.2)

Pour configurer OntoNostra et Jena SDB à utiliser PostgreSQL comme base de données,

- Eteigner l'instance OntoNostra
- Chercher le fichier se nommant **OntoMeaDBConfig.ttl**
- Copier cette configuration (cf, Figure 36 : Configuration base pour OntoMea) en faisant attention au nom de la base, du Host, de l'User et de son mot de passe. Faites attention à bien utiliser le layout2 ici mis en gras.

```
@prefix sdb:    <http://jena.hpl.hp.com/2007/sdb#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ja:     <http://jena.hpl.hp.com/2005/11/Assembler#> .

_:c rdf:type sdb:SDBConnection ;
    sdb:sdbType         "postgresql" ;
    sdb:sdbHost         "OntoNostra" ;
    sdb:sdbName        "OntoNostra" ;
    sdb:sdbUser        "user" ;
    sdb:sdbPassword     "password" ;
    sdb:driver          "org.postgresql.Driver"
;

[] rdf:type sdb:Store ;
    sdb:layout      "layout2/index" ;
    sdb:connection _:c ;
.
```

Figure 36 : Configuration base pour OntoMea

- Ajouter le driver JDBC PostgreSQL-8.2-507.jdbc4.jar dans le répertoire /lib du dossier OntoNostra.
- Redémarrer l'instance OntoNostra

Le rôle du driver est de faire d'intermédiaire entre la base de données, dans le projet : PostgreSQL et un API : SDB. Attention chaque version de chaque base de données à son propre driver JDBC, référez-vous à la page de Jena : http://jena.sourceforge.net/DB/index.html qui indique quel driver pour quelle SGBD. Faite attention chaque version de base peut avoir son propre driver**.**

12.2 KEYSTORE (SECURITE)

La partie la plus laborieuse a été la configuration d'un Keystore et des certificats. Comme expliqué précédemment le Keystore va servir à stocker les certificats qu'OntoNostra va envoyer au client. On va ici s'attarder sur le mode de création d'un Keystore *(cf. Sécurité*). Il existe deux moyens par programmation et par ligne de commande dos. Nous avons utilisé le programme du JDK **Keytool** se trouvant dans le répertoire bin. Le programme Keytool crée et mémorise les clés et les certificats dans un **Keystore**.

La ligne de commande présentée ci-dessous, génère un **fichier** (-genkey) qui a comme **alias** (-alias [nom-alias]) dans un **Keystore** (-Keystore [nomdustockage]) avec un **mot de passe** (-storepass [password]).

Après avoir tapez sur la touche Enter, le Keytool va vous demander d'entrer différentes variables listées de suite (nom et prénom, raison commerciale, nom société qu'on va utiliser comme nom donné au OntoMea, localité, région et le code pays fait de 2 lettres).

Finalement, vous disposez d'un nouveau fichier Keystore avec une extension .jks ou ks se nommant **[nomdustockage].**

```
Keytool –genkey –keyalg RSA –alias [nom-alias] –keystore
[nomdustockage] –storepass [password]
```

Pour avoir un certificat public valide et les utiliser dans nos connexions HTTPS, on peut créer un certificat self signed à l'intérieur du Keystore du serveur. Un certificat self signed est simplement celui dans lequel un issuer et un sujet ont la même entité. Il peut être extrait par le précédent Keystore avec la ligne de commande suivante :

```
keytool –export –keystore [nomdustockage] –alias [nom-alias] –
file [certificiat.cer]  -storepass [password]
```

La création des certificats se fait elle par l'intérmédiaire des Web Services qui se trouvent sur OntoNostra. Le serviice reprend le keystore, stocké dans le serveur central, créé avec les lignes de commandes. Mais il y a aussi la possibilité de créer des certificats par ligne de commande. Référez-vous au site d'OpenSSL http://www.openssl.org/ programme dédié à la sécurité.

12.3 SERVEUR WEB

Etant un projet dédié au Web et à son architecture, l'installation et la configuration du Serveur Web a été très importante. Voici les différentes étapes détaillée pour l'installation et la configuration de Tomcat, Axis2 et ANT. Pour des raisons de praticité, je vais utiliser des abréviations pour ne pas toujours inscrire l'adresse complète.

- %TOMCAT_HOME% = où se trouve dossier Tomcat
- %AXIS2_HOME% = où se trouve dossier Axis2

12.3.1. ANT

Ant comme expliquée à la section 8.4, est très utile pour les projets Java. Il sera beaucoup utilisé durant toutes cette configuration. Ci-dessous, présenter une configuration pour utiliser pleinement Ant et ses fonctions.

- Dézipper le fichier ANT.zip le à la racine du lecteur c :.
- Ajouter les références en écrivant par ligne de commande ce qui est affichée ci-dessous.

Il est nécessaire d'avoir un jdk (Java Development Kit) implémenté sur la machine pour faire tourner ces outils.

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\jdk-1.5.0.05
set PATH=%PATH%;%ANT_HOME%\bin
```

A la fin de la configuration

- Ecrire dans la commande dos « **Ant** ».

Cette commande va chercher si dans le dossier se trouve un **build.xml**. (Build.xml est un fichier de configuration (la carte géographique) pour le projet). Si Ant ne le trouve pas vous aurez aussitôt un message d'erreur, En revanche, si vous recevez un message de variable inconnue, la configuration sera fausse. Site de documentation : http://ant.apache.org/manual/index.html.

12.3.2. TOMCAT

Tomcat avec Apache ayant créé un installeur Windows pour mettre ce programme dans une machine Windows, l'installation n'a pas été très complexe.

- Installer Tomcat et son serveur web intégré Apache grâce à la source **apache-tomcat-6.0.14.exe**.
- Configurer en laissant pour l'instant les variables par défaut. Faites attention à ajouter un mot de passe assez complexe pour la partie manager.
- Pour s'assurer que l'installation de Tomcat s'est bien déroulée. Lancer le fichier **« startup.bat »** qui se trouve dans le dossier bin de %TOMCAT_HOME%.
- Mettez l'adresse Web http://localhost:8080/ . Si vous apercevez l'image ci-dessous (cf. Figure 37 : Page d'accueil Tomcat), cela signifie que tout s'est bien déroulé. L'installation est réussite.

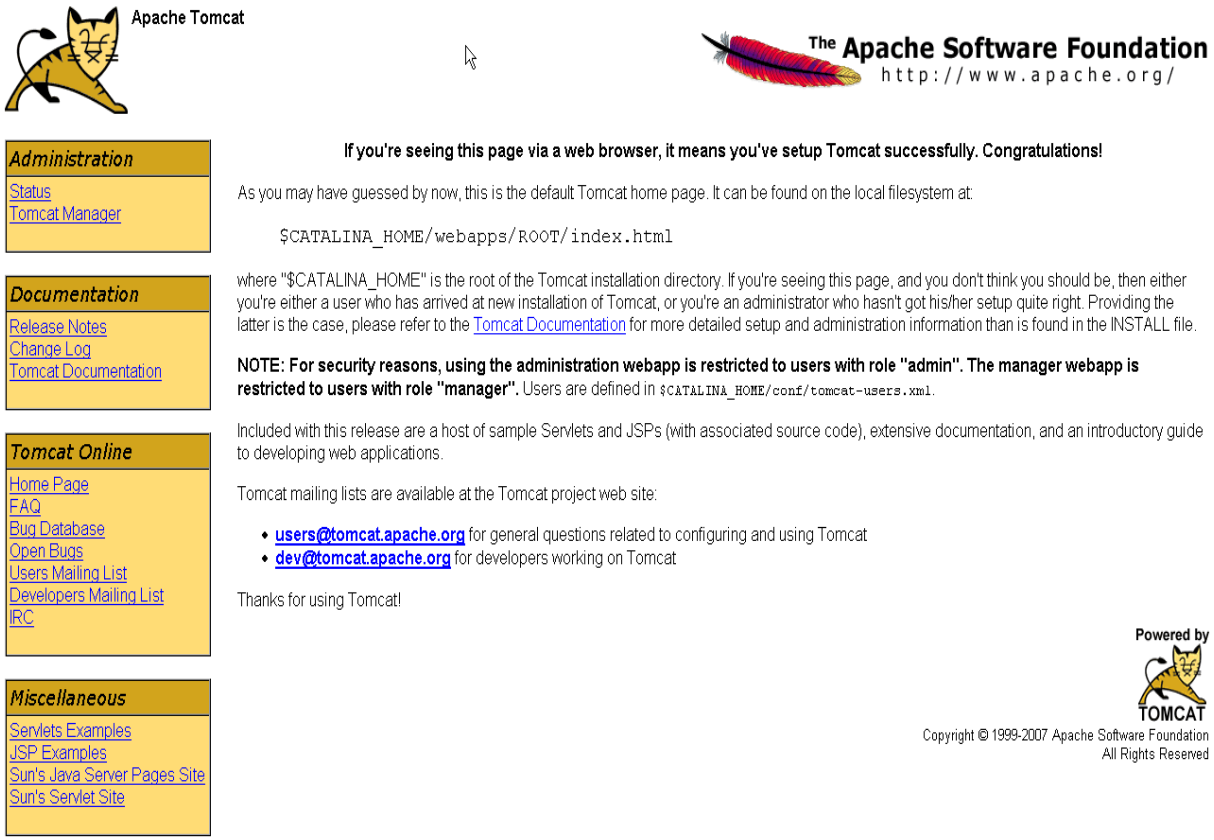


Figure 37 : Page d'accueil Tomcat

HTTPS

- Pour configurer, la partie **https** du serveur web donc la partie sécurisé du serveur Web. Tomcat a besoin d'ajouter des fonctionnalités. La configuration se fait sur le fichier server.xml se trouvant sur **%TOMCAT_HOME%\Conf\server.xml**.
- Ici les lignes à ajouter et à contrôler dans ce fichier.
    - Cherche <Listener> et contrôle si **SSLEngine="ON**"

```
<Listener
className="org.apache.catalina.core.AprLifecycleListener"
SSLEngine="on" />
```

- **Connector Port:** spécifie le numéro du port qu'on veut utiliser pour la connexion http (par défaut 8443)
- **keystoreFile**: mettre le path (chemin) où se trouve le Keystore par exemple dans le dossier de configuration de Tomcat
- **keystorePass**: Indique le mot de passe pour donner la possibilité à Tomcat d'ouvrir le Keystore

```
<Connector
           port="8443" minSpareThreads="5" maxSpareThreads="75"
           enableLookups="true" disableUploadTimeout="true"
           acceptCount="100"  maxThreads="200"
           scheme="https" secure="true" SSLEnabled="true"
          keystoreFile="C:\Program Files\Apache Software
Foundation\Tomcat 6.0\conf\ServerKeyStore"
                 keystorePass="nicola"
           clientAuth="false" sslProtocol="TLS"/>
```

Pour conclure, après avoir relancé le service Tomcat, vous pouvez entrer cette adresse https://localhost:8443/ dans votre browser web et contrôler si la configuration a fonctionné. Suite à l'acceptation du certificat vous aurez la même page d'accueil que http://localhost:8080/ (*cf. Figure 37 : Page d'accueil Tomcat*). Documentation officiel de tomcat en ligne : http://tomcat.apache.org/tomcat-5.5-doc/ssl-howto.html.

12.3.3. AXIS

Dans les sources les configurations sont déjà faites. Axis2 a été configuré pour ajouter la partie sécurité et activer la fonction Rest.

- Télécharger la distribution binaire et copier le dossier Axis2 à la racine du disque.
- Modifier les lignes de configuration, ci dessous dans axis2.xml. Ce fichier sert à configurer Axis et tous les modules intégré. Axis2.xml se trouve dans le dossier %AXIS2_HOME%\conf\Axis2.xml. Ici Ces lignes servent à mettre à disposition :
    - le déploiement des Web Services à chaud (Sans redémarrer le serveur)
    - MTOM (Message Transmission Optimizazion Mechanism) est un nouveau mode de transmission en termes d'échange de données binaire entre service Web et clients de langage différents. Par conséquence laissez échanger des informations entre une application cliente Windows et un Web Service Java.

```
    <parameter name="hotdeployment">true</parameter>
    <parameter name="hotupdate">false</parameter>
    <parameter name="enableMTOM">true</parameter>
    <parameter name="enableSwA">false</parameter>
```

- Ajouter ces trois paramètres, pour donner la possibilité à Axis2 de faire du Mtom. Vous choisissez de laisser la possibilité :
    - de faire du cache « **cacheAttachments** »,
    - sélectionner le dossier où mettre les fichiers « **attachmentdir** »
    - la taille des fichiers « **sizeThreshold** ».

```
  <parameter name="cacheAttachments"
locked="false">true</parameter>
    <parameter name="attachmentDIR" locked="false">temp
directory</parameter>
    <parameter name="sizeThreshold"
locked="false">4000</parameter>
```

Rest

Pour donner la possibilité à Axis de faire du REST, rechercher la ligne, ci-dessous, de configuration disponible dans le fichier **Axis2.xml** et mettre la valeur **false**.

```
    <parameter name="disableREST" locked="true">false</parameter>
```

Rampart

Pour implémenter la sécurité sur Axis, il est nécessaire de lui ajouter un module **Rampart** dédié à la sécurité pour Web Service qui se trouve à cette adresse web : http://ws.apache.org/rampart/.

- Prendre le zip **rampart-1.3.zip** et dézipper-le.
- Copier **rahas-1.3.mar** et **rampart-1.3.mar** dans le dossier se trouvant %AXIS2_HOME%\repository\modules\. Les fichiers .mar sont des modules qui implémentent des fonctions aux Web Services.
- Copier aussi les librairies se trouvant dans le répertoire lib du module dans le répertoire lib d'%AXIS2_HOME%.
- Ajouter la référence à Axis2.xml

```
<module ref="rampart"/>
```

Distribution

A la fin de la configuration, il faut compiler Axis2 pour créer un fichier War e l'ajouter Axis au serveur Tomcat. Le fichier WAR est un fichier jar pour les web application.

- Se positionner, par ligne de commande, dans le dossier Webapp se trouvant dans Axis
- Taper la ligne de commande ANT dans le dossier %AXIS_HOME%\webapp. Dans ce fichier se trouve un fichier build.xml qui va compiler Axis et créer le fichier War dans le dossier %AXIS_HOME%\dist

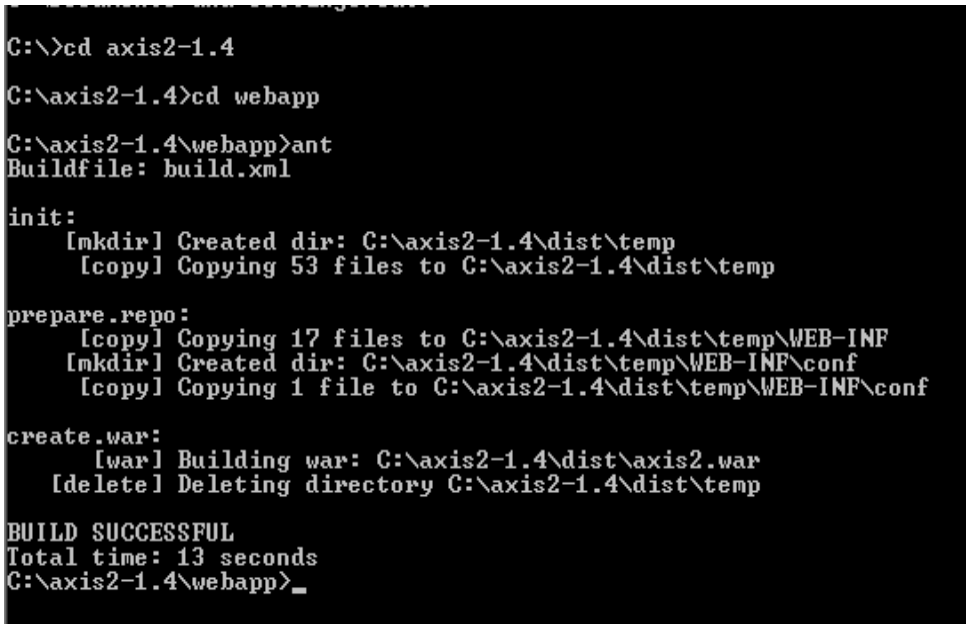


Figure 38 : Création War Axis

- Dès la création du fichier, ajouter le fichier WAR dans le dossier %TOMCAT_HOME%\webapps\ ou passer par la partie Web de Tomcat et le menu Manager qui contient un browser pour aller chercher où se trouve le fichier War.
- Pour contrôler la réussite de l'installation d'Axis2 et son bon déroulement. Redémarrer Tomcat et écrire dans le browser http://localhost:8080/axis2.

Si on aperçoit l'image (cf. Figure 39) cela signifie que tout c'est bien déroule. Il existera dés lors trois liens mis à disposition par Axis :

- **Services** : Liste les services à disposition d'Axis
- **Validate** : Contrôle si les librairies nécessaires au fonctionnement sont disponibles à Axis
- **Administration** : Console d'administration (par défaut le login admin et mot de passe axis2) qui donne la possibilité d'uploader (ajouter) des Web Services, désactiver les services, leur rajouter des modules…

Figure 39 : Page d'accueil Axis2

Ajouter des Web Services à AXIS

- Créer les fichiers aar avec les outils cités à la section 0. Les fichiers aar ont la même fonction de Jar et War. Il s'agit de fichiers compilés par Java expressément utilisés pour les Web Services.
- Copier les fichiers aar créés précédemment, dans le dossier %TOMCAT_HOME%\webapps\Axis2\WEB-INF\services.
- Si les services Tomcat et Axis seront démarrés, ils seront uploadés automatiquement ou alors au prochain redémarrage.
- Voilà les Web Services à votre disposition. Pour accéder aux différents Wev Service http://nomdomain:8080/Axisw/nomService.wsdl. Nom des services proposé dans ce projet :
    - **webservice_Certificat** (Envoi le fichier certificats aux clients et l'ajoute dans le Keystore)
    - **webservice_Envoi** (Liste et renvoie aux clients les fichiers stockés dans OntoNostra)
    - **webservice_Recevoir** (Reçoit les fichiers ou les métadonnées qu'un client désire envoyer)
    - **webservice_Recherche** (Renvoie la liste de tous les URI des fichiers qui font référence à une ressource)

Faite attention à modifier le fichier WSDL pour qu'il pointe à la bonne adresse IP, par défaut « localhost » est créé. Documentatin officiel axis : **http://ws.apache.org/axis/java/index.html**

12.3.4. WSE 3.0 ET VISUAL STUDIO

Ce n'est pas tout d'activer MTOM dans les Web Services, il faut aussi l'activer dans l'application de démonstration d'OntoMea ou dans toutes autres applications qui veulent utiliser ces Web Services.

Pour laisser interagir des Web Services Java en Mtom et des applications clientes Windows et vice-versa, il faut installer WSE 3.0 de Microsoft (Web Service Enhancements) suit Wse 1.0 et WSE 2.0, il ajoute aussi des nouvelles fonctionnalités à Visual Studio 2005. WSE 3.0 donne la possibilité d'utiliser le standard MTOM de W3C. Il est utilisé pour faciliter de beaucoup la configuration de la sécurité, du routage. Il trace aussi les messages envoyés entre clients/serveur et à plein d'autres utilitaires très intéressant de diagnostique de problème WSE 3.0 a une incrémentation de performance du 20% des précédents. Téléchargez-le à http://www.microsoft.com/downloads/details.aspx?FamilyID=018a09fd-3a74-43c5-8ec1-8d789091255d&displaylang=en.

Après l'installation, il existera un nouvel onglet dans Visual Studio pour mettre en place le WSE 3.0. Click droite sur le projet dans lequel activer MTOM et au fond vous apercevrez un onglet WSE Setting 3.0. Si vous cochez « Enable this projet for WSE » (cf. Figure 40 : Wse 3.0), dès lors mtom sera activé dans le projet. Cette mini application va modifier ou créer un fichier de configuration (web.config ou app.config) et ajouter les références pour activer cette nouvelle technologie.

Ajouter dans la partie « Messaging » Client Mode « on » pour mettre dans le fichier que l'application a le rôle de client et va recevoir des fichiers. Pour avoir un exemple d'utilisation : http://www.dotnet-news.com/lien.aspx?ID=29305

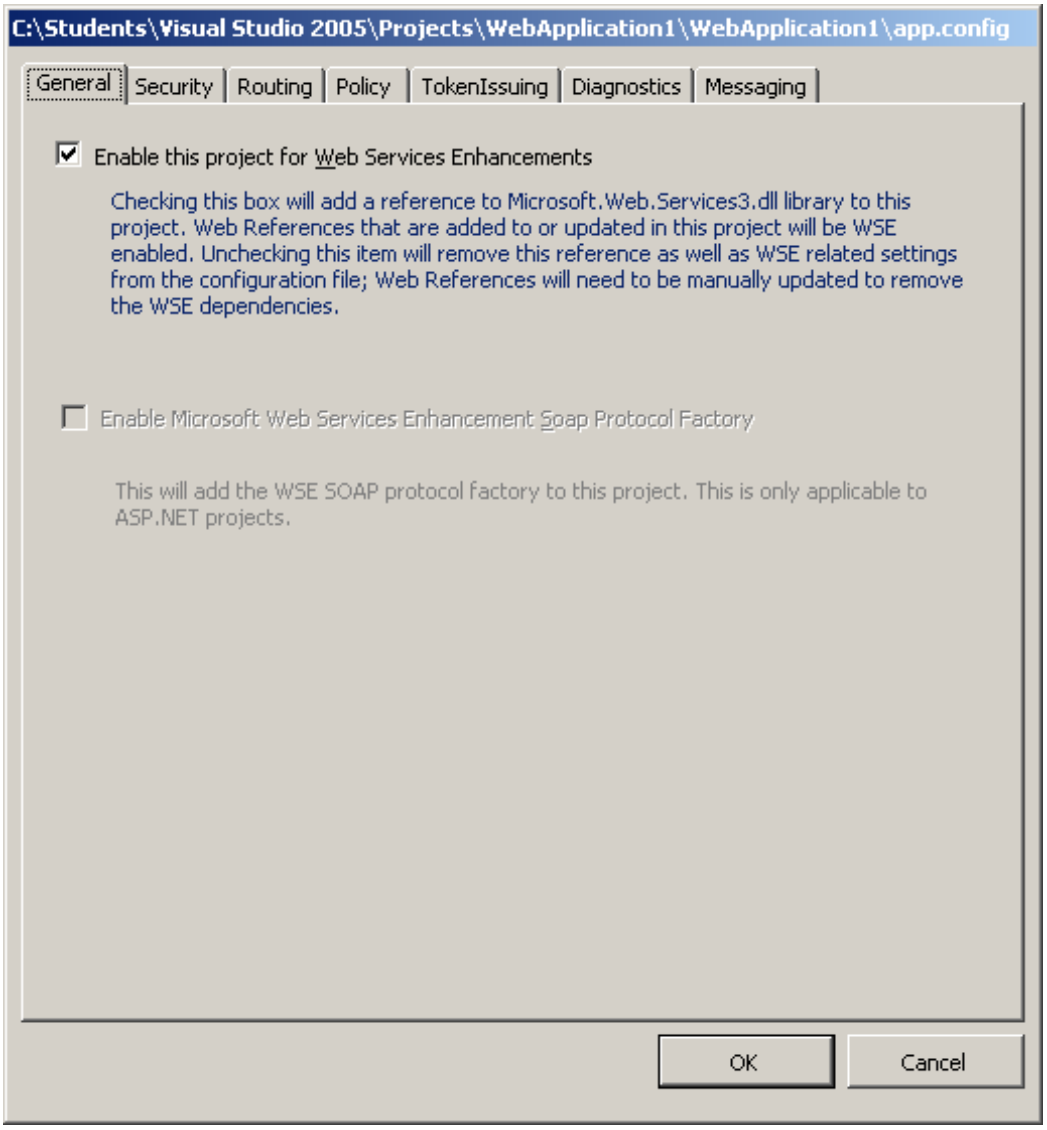


Figure 40 : Wse 3.0

13. CONCLUSION

Pour conclure, la partie essentielle de mon travail de diplôme a été un grand effort de recherche. Il fallait trouver les meilleures solutions adaptées au projet et au Monde Web 3.0 donc le projet a été comme même très téhorique et très tourné vers la recherche. En finale, l'une des seules parties concrètes du Travail a été la création des Web Services qui offrent maintenant aux OntoMea d'accéder aux fonctions d'OntoNostra par l'intermédiaire d'une authentification des serveurs locaux et aussi des utilisateurs.

L'utilisation de Rest dans notre style d'architecture laisse une très grande marge d'évolution et je suis sûre que celle-ci est la plus proche aux besoins du Web Sémantique. L'identification des URI sera beaucoup plus facile à mettre en place grâce à REST et ses modèles et il pourra suivre l'évolution de l'informatique. Une autre solution adaptée, l'OpenID, est une nouvelle façon de s'identifier dans le monde du Web et inséré dans le monde Web 3.0. Son intégration avec FOAF le rend facilement utilisable par OntoMea. Le reste du travail n'intervient pas directement dans le Web 3.0 et dans sa problématique mais devait, malgré tout, faire attention à ses prérogatives.

Une **évolution** possible pour le projet Memoria-Mea est d'ajouter deux autres idées innovantes qui viennent du Web 3.0 : OAuth et Apml (cf. Figure 41).

OAuth (Open Authentification) est un protocole d'authentification. Il reprend le concept d'OpenID. Néanmoins, comme son nom l'indique il est dédié à l'authentification. OAuth fonctionne sur la base suivant laquelle un utilisateur peut déléguer aux applications ou aux sites Web et accéder aux données sauvegardées (par exemple un site comme Flickr). En continuant à utiliser cet exemple, l'utilisateur pourra donc donner le droit à une application d'utiliser les photos et les Web Services, ici de Flickr.

*« **Apml** (Attention Profiling Mark-up Language) est un format basé sur le XML visant à décrire les centres d'intérêts et de désintérêts d'une personne. APML permet à ses utilisateurs de partager leur profil d'attention de manière similaire à ce que l'OPML permet de faire avec la lecture de flux RSS. L'idée derrière APML est de compresser toutes les données d'attention au sein d'un fichier unique qui regrouperait ainsi tous les centres d'intérêts de son utilisateur. » [43]*

Vous remarquerez sûrement dans l'image, que trois des cinq principales technologies typique du web 3.0 sont déjà intégré à OntoNostra et OntoMea. :

- OpenID (identification section 6.1)
- Foaf (représentation section 3.2)
- RDF (stockage section 3.1 et 9.2)

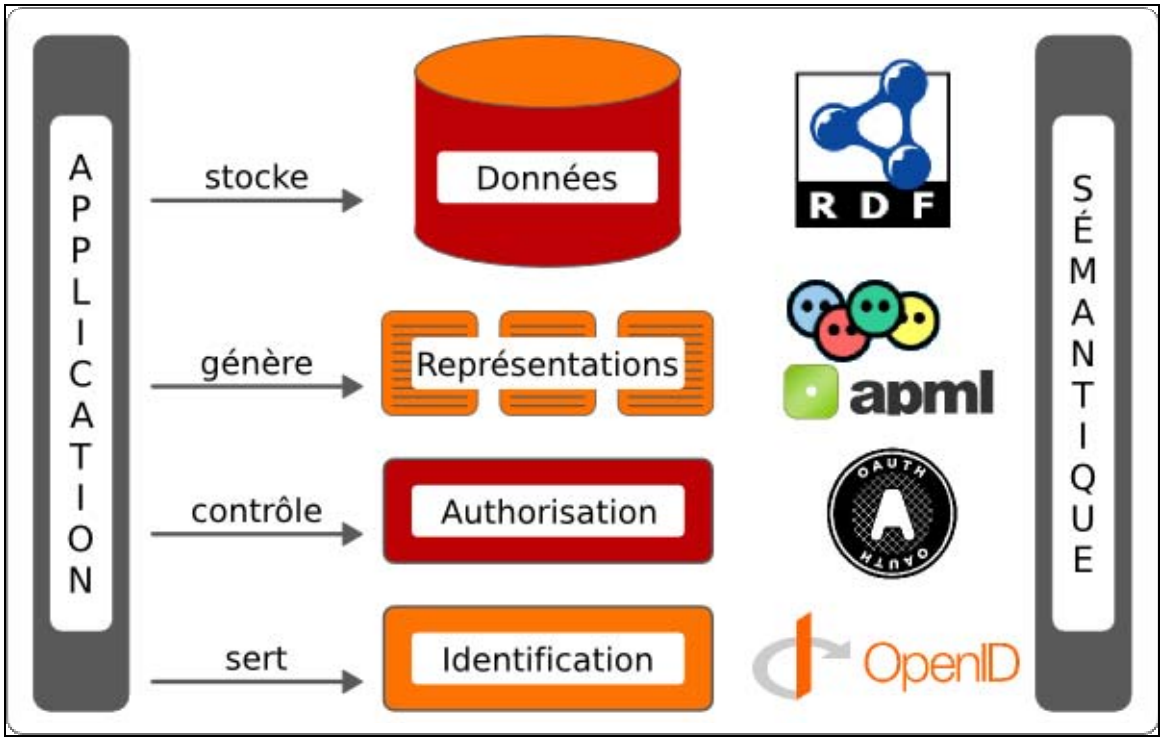


Figure 41 : Evolution Web 3.0 [42]

D'un point de vue personnel, le fait de travailler avec de nouvelles technologies m'a permis de découvrir de nombreuses solutions, des nouveaux outils de développement et des architectures différentes et originales. Les voici listées : MTOM, AXIS2, Netbeans, OpenID, Glassfish, Tomcat, Web Service, Rest… pour voir un aperçu (*cf. Figure 42 : Outils utilisés*). La découverte et l'approfondissement de nouvelles technologies et même les milles tentatives non-abouties m'ont apporté une nouvelle vision de l'informatique et de la gestion d'un projet. Les nombreuses recherches effectuées ont été enrichissantes bien que la difficulté majeure a été de trouver de la documentation de qualité et compréhensible dans les milliers de pages web visitées.

Le Web sémantique étant une technologie récente, il n'existe pas encore une réelle méthodologie pour le traitement de telles informations. Certainement, le Web 3.0 étant une nouveauté en plein essor, il fallait suivre l'évolution et l'apparition de nouvelles technologies. La phase d'analyse a été par conséquence extrêmement importante afin de dégrossir les différentes solutions émergeantes et du suivi de ces solutions. Outre le Web sémantique lui-même, certaines étapes de ce travail ont concerné la plupart des domaines de l'informatique modernes tels que les bases de données, le réseau, le développement d'application web et d'application windows, la sécurité, OpenID, l'intéropérabilité entre différentes plateformes ou encore l'architecture web.

Pour finir, je souhaite mettre en évidence les excellentes conditions de travail et je tiens à remercier vivement mon professeur chargé du suivi **Anne Le Calvé** ainsi que mon responsable technique **Fabian Cretton** pour leur disponibilité, leurs conseils et leur aide morale et technique tout au long de ce travail.

Figure 42 : Outils utilisés

14. DECLARATION SUR L'HONNEUR

Je déclare, par ce document, que j'ai effectué le travail de diplôme ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de diplôme, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Fabian Cretton, Assistant, Institut Informatique de gestion, HES-SO Valais Wallis
- Anne Le Calvé, Professeur, Institut Informatique de gestion, HES-SO Valais Wallis

Signature :

15. GLOSSAIRE

ANT	Logiciel qui automatise toutes les tâches répétitives dédiées au langage Java, il fait le travail d'une carte géographique qui montre où aller pour trouver ces informations.
API	Une API (Application Programming interface) permet de définir la communication entre différents composants informatique
Assertion	Une expression qui suit une grammaire spécifique chi nomme une ressource spécifique, une propriété (attribut) et fourni la valeur de cette propriété pour cette ressource.
Axis 2.0	Framework d'Apache écrit en Java dédié au Web Service
Buoncy Castle	Bouncy Castle est une bibliothèque de cryptographie libre et open-source
Certificat	« Un certificat électronique est une carte d'identité numérique dont l'objet est d'identifier. L'autorité de certification fait foi de tiers de confiance et atteste du lien entre l'identité physique et l'entité numérique. Le standard le plus utilisé pour les certificats numériques est le X.509. » (Source Wikipedia)
Checksum	Le checksum d'un fichier est une séquence de chiffres et de lettres définissant de manière précise (mais non unique) un fichier afin de savoir si il a été altéré.
Client/serveur	Les clients sont des machines qui cherchent à exploiter des services comme proposés par un serveur
CRUD	« Acronyme informatique anglais souvent utilisé lorsqu'on parle du langage de requête SQL. Ce sont les quatre opérations de base qui constituent tout ce qu'il est possible de faire sur un SGBD. Create, Read (ou Retrive), Update, Delete » (Source Wikipedia)
D2RServeur	D2R Server est un outil Open Source utilisé pour publié le contenu d'un base de donnée relationnel dans le Web Sémantique en fichier du type RDF

Dbpedia	Dbpedia est un projet dont le but est d'extraire les informations présentes dans Wikipedia et de les transformer dans un format structuré/normalisé (RDF) en suivant une ontologie établie pour chaque type d'objet
Driver	Le rôle du driver est de faire d'intermédiaire entre une base de données et un API.
FOAF	Friend of a Friend projet du Web sémantique dédié à la personne
GeoNames	GeoNames est une base de données Sémantique dédié à la géographie gratuite et accessible par internet qui offre des Web Services disponible pour tous et qui renvoient les différents lieux. Il fonctionne come un Wiki donc les utilisateurs sont libres d'ajouter ou améliorer les données.
HTTP	« Le HyperText Transfer Protocol, plus connu sous l'abréviation HTTP est un protocole de communication client-serveur développé pour le World Wide Web » (Source Wikipedia)
Inférences	« L'inférence est une opération mentale qui consiste à tirer une conclusion (d'une série de propositions reconnues pour vraies). Ces conclusions sont tirées à partir de règles de base. » (Source Wikipedia)
JENA	Framework Java dédié au Web Sémantique implémenté de plusieurs API (SDB, ARQ, Joseki, Eyeball…)
Keystore	Une mini base de donnée qui contient les clés publiques de plusieurs certificats
Keytool	Outil pour la génération, l'importation, et la gestion de clefs et de certificats contenus dans les magasins de clefs
Lien	Une relation entre deux ressources quand une ressource (une représentation) se réfère à une autre ressource au moyen d'un URI.
Maven	Maven est un autre outil du monde apache qui ressemble beaucoup à ant. La grande différence de Ant, ou vous décrivez les taches à réaliser, dans Maven vous décrivez votre projet, et lui connait les taches à réaliser. C'est une approche différente, amenant l'utilisation de fichiers de configurations plus légers
Memoria Mea	Memoria-Mea est un projet de gestion de données personnelles composé par une multitude de modules distincts.

Métadonnées	C'est des données qui décrivent des données et grâce à cela les machines puissent les travailler
MTOM	Message Transmission Optimizazion Mechanism est un mode de transmission en termes d'échange de données binaire entre service Web et clients de language différents.
NAMED Graph	Un Triple Store permettant de stocker les différents graphes séparément en leur donnant un nom.
Oauth	OAuth (Open Authentification) est un protocole d'authentificationdu type d'OpenID. OAuth fonctionne sur la base duquel un utilisateur peut déléguer aux applications ou aux sites Web à accéder aux données sauvegarder
Ontologie	« En informatique et en science de l'information, une ontologie est un ensemble structuré de concepts permettant de donner un sens aux informations. Elle est aussi un modèle de données qui représente un ensemble de concepts dans un domaine et les rapports entre ces concepts. Elle est employée pour raisonner au sujet des objets dans ce domaine. » (Source Wikipedia)
OntoMea	OntoMea est un prototype de moteur sémantique permettant de gérer une base de connaissances et d'effectuer des déductions de nouvelles informations grâce à un moteur d'inférences intégré
OpenID	un système d'authentification décentralisé qui permet le partage d'attributs
OWL	Web Ontology Language est en effet un langage basé sur RDF qui permet de définir des ontologies structurées
P2P	Modèle de communication, dans lequel chacune des machines peuvent avoir le rôle une fois du serveur une fois du client.
Protégé	Protégé est un éditeur d'ontologies Open Source. La plateforme Protégé permet la création et l'édition d'ontologies
Protocole	Protocole est une norme qui permet à deux machines de se comprendre et de parler la même « langue » même si c'est des machines différentes
Rampart	Rampart est un module dédié à la sécurité pour Web Service

RDF	RDF est un langage généraliste basé sur XML dont le but est d'offrir une représentation des informations contenues sur Internet.
Ressource	Un objet abstrait qui représente un objet physique comme une personne, une image ou un livre.
REST	*REpresentational State Transfer* est un style d'architecture orienté ressource (ROA) pour des systèmes distribués crée par Fielding
SIMILE	Ce projet web du MIT crée et développe de nombreux utilitaires Open Source expressément dédiés au Web Sémantique.
Sparql	Langage de requête pour données et métadonnées dédié au Web sémantique
Soap	Simple Object Access Protocol est un autre protocole sur XML, pour l'échange de messages entre différents composants software.
Svn	Système de gestion de versions, il agit sur une arborescence de fichiers afin de conserver les versions des fichiers
SSL	rotocole SSL se base sur une infrastructure à clé publique (PKI) our la double fonction d'authentification des intervenants et ermettre les échanges sûrs entre eux.
Tomcat Apache	Tomcat est un serveur d'application dédié à Java intégré à un serveur Web Apache.
Triplets	Une représentation d'une assertion utilisé par RDF, composé par la propriété, de l'identifiant de la ressource e la valeur de la propriété. Synonyme : Statement
Uri	Uniform Ressource identifier. Identificateur unique à travers le Web, à ne pas confondre avec Url (Uniform Ressource Locator) qui est « l'adresse » où se trouve l'Uri
URLRewriter	URLRewriter utilise une fonction de réécriture configurable par règles pour réécrire au vol des URL et pour leur donner un aspect plus compréhensible.
Web	Un espace d'informations dans lequel des éléments dignes d'intérêt sont identifiés par des URI.

Web sémantique	« Le Web sémantique désigne un ensemble de technologies visant à rendre le contenu des ressources du World Wide Web accessible et utilisable par les programmes et agents logiciels, grâce à un système de métadonnées formelles, utilisant notamment la famille de langages développés par le W3C » (Source Wikipedia)
Web Service	Web Service sont des composants métiers qui s'exécutent à travers l'internet ou l'intranet, auto-suffisants et auto-descriptifs (dans le sens où toute la description se trouve avec le travail proposé). L'interopérabilité entre différents agents est le travail principal des Web Services.
WSE 3.0	Web Service Enhancements 3.0 offre la possibilité d'utiliser le nouveau standard MTOM.
WSIT	Web Services Interoperability Technologies (WSIT) permet l'interopérabilité entre la plateforme Java et Windows Communication Foundation (WCF)
XLS	eXtensible Stylesheet Language qui transforme un résultat XML/RDF dans un format plus facile à lire, un css dédié au XML
XML	Extensible Markup Language. À la base, XML est un langage de formatage de documents recommandé par le W3C, consortium chargé de développer les protocoles de l'Internet.
XMLNS	XMLNS (XML namespace). Ce qui signifie que le document XML et ses variables utilisent les noms de ces namespaces identifié par leur URI.

16. INDEX

16.1 BIBLIOGRAPHIE

[1] **Web Sémantique.ch,** *site Suisse dédié au Web sémantique*, http://www.websemantique.ch/index.php

[2] **Memoria-Mea***, projet global comprenant OntoMea et OntoNostra,* http://memoria-mea.hefr.ch/,

[3] **W3C**, *World Wide Web Consortium,* http://www.w3.org/, (consulté le 15.09.2007)

[4] **RDF**, *Site dédié à RDF du W3C*, http://www.w3.org/RDF/, (consulté le 15.09.2007)

[5] **FOAF**, *Site officiel du projet FOAF*, http://www.foaf-project.org/, (conulté 30.09.2007)

[6] **SPARQL**, *Site dédié à SPARQL du W3C*, http://www.w3.org/2001/sw/DataAccess/, (consulté 17.09.2007)

[7] **OWL**, *Site dédié à SPARQL du W3C et traduction,* http://www.w3.org/2004/OWL/, http://www.yoyodesign.org/doc/w3c/owl-guide-20040210/#Introduction, (consulté 18.09.2007)

[8] **Jena**, *Semantic Web Framework*, http://jena.sourceforge.net/, (consulté 15.09.2007)

[9] **Jena Forum**, *Forum développeur dédié à Jena,* http://tech.groups.yahoo.com/group/jena-dev/, (consulté le 20.09.2007)

[10] **Protégé**, Ontologie éditeur, http://protege.stanford.edu/, (consulté 12.09.2007)

[11] **Clever-Age.com**, *le web sémantique en entreprise : comment et à quels niveaux ?* http://www.clever-age.com/spip.php?page=article&id_article=498 (consulté le 21.03.2008).

[12] **Epfl.ch,** *Peer-to-peer or not,* http://ditwww.epfl.ch/SIC/SA/SPIP/Publications/spip.php?article928 (consulté le 15.06.2008).

[13] **Wikipedia.fr,** *SMTP***,** http://fr.wikipedia.org/wiki/SMTP (consulté le 01.06.2008)

[14] **Développez.com**, *Etude et utilisation des technologies des P2P*, http://schuler.developpez.com/articles/p2p/ (consulté 10.04.2008)

[15] **Wikipedia.fr**, *REST*, http://fr.wikipedia.org/wiki/Representational_state_transfer

[16] **Wikipedia.fr**, *HTTP,* http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP, (consulté le 01.12.2007)

[17] **Figer.com,** *REST, un style d'architecture universel,* http://www.figer.com/Publications/REST.htm. (consulté le 09.10.2007)

[18] **W3.org Architecture Web**, *Architecture of the World Wide Web, Volume One,* http://www.w3.org/2001/tag/webarch/ (consulté le 3.10.2007)

[19] **Wiwiss.fu-berlin.de**, *How to Publish Linked Data on the Web*, http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/ (consulté le 2.06.2008)

[20] **Dicodunet.com**. *définition du Checksum*, http://www.dicodunet.com/definitions/google/checksum.htm (consulté 7.05.2008)

[21] **Geonames**, *Site sémantique dédié aux lieux géographique*, http://www.geonames.org, (consulté 01.10.2007)

[22] **Dbpedia**, *Encyclopédie RDF*, http://dbpedia.org/About, (consulté le 02.11.2007)

[23] **SIMILE**, *Semantic Interoperability of Metadata and Information in un Like Environments*, http://simile.mit.edu., (consulté 01.11.2007)

[24] **OpenID,** *Site Officiel,* http://Openid.net (consulté 01.12.2007)

[25] **Wikipedia.fr**, *OpenID*, http://fr.wikipedia.org/wiki/OpenID, (consulté le 01.12.2007)

[26] **MyOpenID**, *Serveur d'identité d'OpenID*, www.myOpenID.com., (consulté le 01.12.2007)

[27] **OpenIDDirectory**, *Liste de tout les sites acceptant OpenID comme authentification*, http://openiddirectory.com/, (consulté 02.02.2008)

[28] **JDN développeurs**, *l'identité numérique avec OpenID*, http://www.journaldunet.com/developpeur/tutoriel/theo/070117-openid.shtml (consulté le 02.12.2007)

[29] **Tomcat**, *Web Serveur*, http://tomcat.apache.org/

[30] **Axis**, *Container Web Service de Tomcat Apache*, http://ws.apache.org/axis2/index.html

[31] **Rampart**, *Module dédié à la sécurité* d'Axis, http://ws.apache.org/rampart/,

[32] **ANT**, *outil inclus dans le monde Tomcat*, http://ant.apache.org/,

[33] **SVN**, *Site officiel*, http://subversion.tigris.org/,

[34] **Elamb.org**, *Security Now Episode #95* http://elamb.org/category/certificationsecuritygeneral-security-conceptsaccess-control-models/ (consulté le 15.05.2008)

[35] **Biologeek.com**, *Le web de demain* http://www.biologeek.com/journal/index.php/Web-semantique (consulté le 14.02.2008)

[36] **Wiwiss.fu-berlin.de**, *D2RServer*, http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/ (consulté 20.11.2008)

[37] **PostgreSQL**, *Site officiel en français dédié à PostgreSQL*, http://www.postgresqlfr.org/, (consulté 12.12.2007)

[38] **Mokabyte.it,** *Accès HTTPS avec JAVA*, http://www2.mokabyte.it/cms/article.run?articleId=CVQ-VUZ-HSD-H3A_7f000001_18073931_3393f5f0 (consulté le 20.03.2008)

[39] **Eclipse**, *IDE dédié à Java,* http://www.eclipse.org/

[40] **Netbeans,** *IDE dédié à Java et à Ruby*, http://www.netbeans.org/,

[41] **Visual Studio**, *IDE dédié à Microsoft*, http://www.microsoft.com/france/msdn/vstudio/default.mspx,

[42] **Biologeek**, *Ma Killer app pour le Web Sémantique,* http://www.biologeek.com/web-semantique/ma-killer-app-pour-le-web-semantique/ (consulté 12.01.2008)

[43] **Wikipedia,** *APML*, http://fr.wikipedia.org/wiki/APML

[44] **Bouncy Castle,** *Librairie dédié à la sécurité Open Source*, http://www.bouncycastle.org/

[45] **OpenSSL,** *Programme dédié à la certificiation et sécurité*, http://www.openssl.org/

[46] **Planète Web Sémantique,** Site dédié au Web Sémantique et à ses applications, http://planete.websemantique.org/

[47] **WSE 3.0,** Web Service Enhancements, http://www.microsoft.com/downloads/details.aspx?FamilyID=018a09fd-3a74-43c5-8ec1-8d789091255d&displaylang=en.

[48] **Rest- Granges.net,** Apprendre REST - un style d'architecture du Web, http://www.la-grange.net/2005/05/rest/

Livre

RESTful Web Services by Leonard Richardson (Author), Sam Ruby (Author),

16.2 TABLE DES ILLUSTRATIONS

16.3 TABLE DES TABLEAUX

17. LISTE DES ANNEXES

17.1 CAHIER DES CHARGES

17.2 FEUILLES MS PROJECT

17.3 FEUILLES DES HEURES

17.4 COOLURIS

17.5 REST TRADUCTION

Cahier des charges

Francesco Nicola De Palma
609_3

Table des matières

1. Introduction

« Nous allons d'un web de documents connectés à un web de données connectées »

Alors que le web 2.0 continue sa mise en place, tous les grands acteurs du web travaillent actuellement sur les nouvelles technologies nécessaires pour permettre aux données de prendre un sens dans le web : le web 3.0. Des ontologies sont créées dans de nombreux domaines afin de définir un vocabulaire permettant d'identifier les données et de décrire leurs relations entre elles.

La puissance et l'enjeu de ce web sémantique viennent de la mise en relation de la multitude de données existantes, et de leur valorisation par la déduction de nouvelles données et de nouvelles connexions. Sur la base de ces inférences, de nouvelles applications aux possibilités semble-t-il infinies pourront voir le jour.

Ce projet ira vers la direction du **déploiement du web 3.0** : c'est le partage d'informations personnelles depuis des sources locales vers un système central.

2. Contexte du travail de diplôme

2.1. PIM (Personal Information Management)

C'est un logiciel d'application qui a comme but d'être un organisateur personnel. Comme outil de gestion de l'information, le but d'un PIM est de faciliter l'enregistrement, le cheminement, et la gestion de certains types « d'information personnelle » (par exemple : Notes personnelles, carnet d'adresse, anniversaire, rendez-vous, e-mail et bien d'autres encore).

Les produits de logiciel de certains PIM sont capables de synchroniser des données avec un autre PIM au-dessus d'un réseau informatique (réseaux ads- mobiles y compris). Ce dispositif habituellement ne tient pas compte des mises à jour continues de données, mais permet plutôt les mises à jour personnelles entre différents ordinateurs, y compris des ordinateurs de bureau, des ordinateurs portables, et des aides numériques personnels.

2.2. Web sémantique

2.2.1. Définition

Qu'est-ce que le web sémantique ? Voilà une définition donnée par l'un des co-inventeurs du web et le premier à avoir donné les principes du web sémantique **Tim Berners-Lee** (patron du W3C **(World Wide Web Consortium))**

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation".

Et de suite la traduction : *"Le Web sémantique est une prolongation du Web actuel qui permet une définition non ambiguë de l'information pour favoriser une meilleure coopération entre les ordinateurs et les personnes ".*

Allons plus loin dans l'explication, « le Web sémantique est entièrement fondé sur le Web actuel et ne remet pas en cause ce dernier. Le Web sémantique se base donc sur la fonction basique du Web « classique » : un moyen de publier et consulter des documents de toutes sortes (texte, image, vidéo, …). Mais les documents traités par le Web sémantique contiennent non pas des textes en langage naturel (français, espagnol, chinois, etc.) mais des informations formalisées pour être traitées automatiquement. Ces documents sont générés, traités, échangés par des logiciels. Ces logiciels (qu'on appel aussi souvent agents) permettent souvent, sans connaissance informatique, de :

- *générer des données sémantiques à partir de la saisie d'information par les utilisateurs ;*
- *agréger des données sémantiques afin d'être publiées ou traitées ;*
- *publier des données sémantiques avec une mise en forme personnalisée ou spécialisée ;*
- *échanger automatiquement des données en fonction de leurs relations sémantiques ;*
- *générer des données sémantiques automatiquement, sans saisie humaine, à partir de règles d'inférences. »*

(http://fr.wikipedia.org/wiki/Web_s%C3%A9mantique)

2.2.2. RDF (Ressource Description Framework)

L'évolution du web sémantique commence par la définition, de la part du W3C, du standard RDF, une application particulière XML qui standardise la définition de relation entre l'information en s'inspirant à la logique des prédicats (l'information s'exprime avec des affirmations qui constitue des triples formés par sujet, verbe et objet).

Par exemple : 1. L'Hes-So a été construite à Sierre.
2. Sierre se trouve en Suisse.

Nous pouvons les schématisé avec le tableau si dessous

	Affirmation 1	Affirmation 2
Sujet:	L'Hes-So	Sierre
Verbe:	a été construite	Se trouve
Objet:	Sierre	Suisse

Alors pour quelques un de ces élément nous pouvons trouver arbitrairement des Web URI (**Uniform Resource Identifier** qui identifie une ressource générique qui peut être une adresse Web, un document, une image, un service ou un Url …).Ci dessous un graphe avec ses URI et le texte en RDF/XML

L'Hes-So http://www.hes-so.ch/
Sierre http://sws.geonames.org/2658606/
Suisse http://sws.geonames.org/2658434/

```
<?xml version='1.0'?>
  <rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     <rdf:Description
        < HesSo >
           http://www.hes-so.ch/
        </ HesSo >
        <Sierre>
           http://sws.geonames.org/2658606/
        </Sierre>
        <Suisse>
           http://sws.geonames.org/2658434/
        </Suisse>
     </rdf:Description>
  </rdf:RDF>
```

2.2.3. OWL (Web Ontologie Langage)

« Voilà une définition d'OWL — est un dialecte XML basé sur une syntaxe RDF. Il fournit les moyens pour définir des ontologies Web structurées.

Le langage OWL est basé sur la recherche effectuée dans le domaine de la logique de description. OWL peut être vu en quelque sorte comme un format de fichier pour certaines logiques de description. OWL permet de décrire des ontologies, c'est-à-dire qu'il permet de définir des terminologies pour décrire des domaines concrets. Une terminologie se constitue de concepts et de propriétés (aussi appelés ***rôles*** *en logiques de description). Un domaine se compose d'instance de concepts.*

En pratique, le langage OWL est conçu comme une extension de RDF et RDF Schéma ; OWL est destiné à la description de classes (par des constructeurs) et de types de propriétés. De ce fait, il est plus expressif que RDF et RDFS, auxquels certains reprochent une insuffisance d'expressivité due à la seule définition des relations entre objets par des assertions. OWL apporte aussi une meilleure intégration, une évolution, un partage et une inférence plus facile des ontologies.

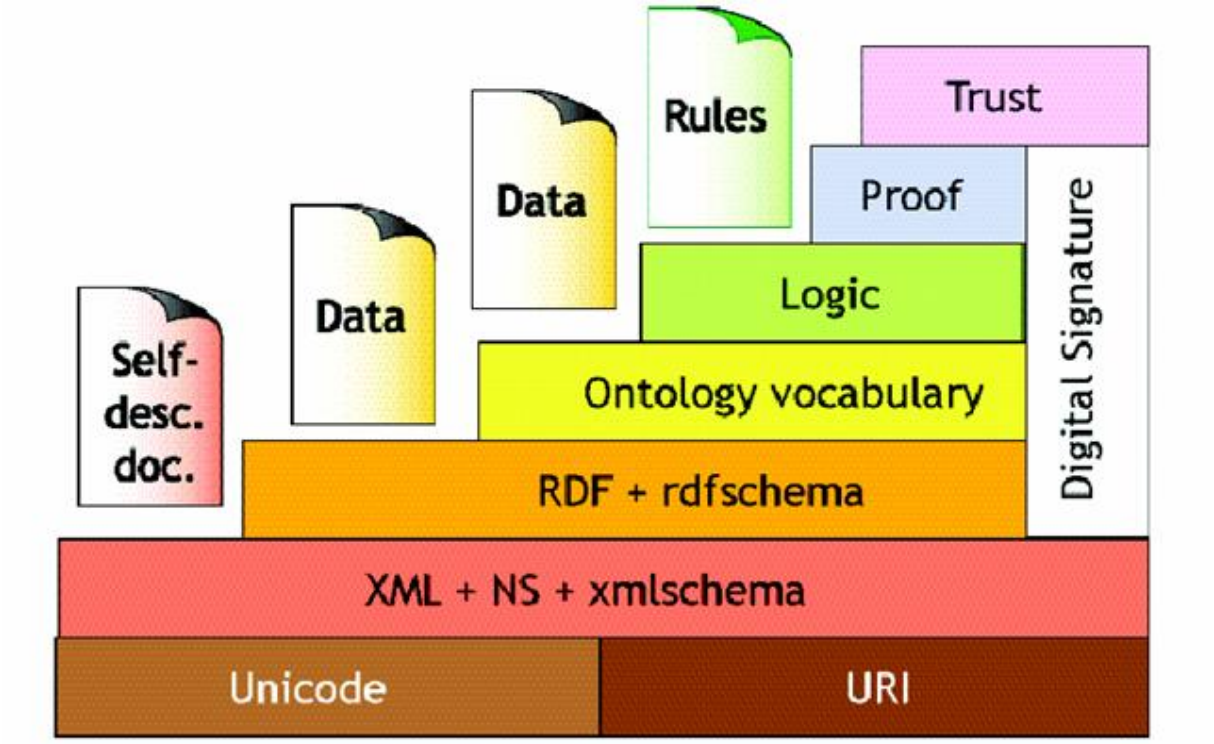
Aux concepts de classe, de ressource, de littéral et de propriétés des sous-classes, de sous propriétés, de champs de valeurs et de domaines d'application déjà présents dans RDFS, OWL ajoute les concepts de classes équivalentes, de propriété équivalente, d'égalité de deux ressources, de leurs différences, du contraire, de symétrie et de cardinalité...

OWL permet, grâce à sa sémantique formelle basée sur une fondation logique largement étudiée, de définir des associations plus complexes des ressources ainsi que les propriétés de leurs classes respectives. OWL définit trois sous langages, du moins expressif au plus expressif : ***OWL-Lite****,* ***OWL-DL*** *et* ***OWL-Full****. Des algorithmes décidables existent pour la totalité d'OWL-Lite. Quoique les problèmes d'inférence d'OWL-DL puissent être résolus en temps exponentiel de façon générale, le comportement est souvent satisfaisant. Il n'existe aucun algorithme d'inférence décidable pour OWL-Full.*

L'OWL est adéquat pour le Web sémantique, car il offre une syntaxe définie strictement, une sémantique définie strictement et selon le niveau peut permettre des raisonnements automatisés sur les inférences et conclusions des connaissances. Les langages sur lesquels il est construit sont largement interprétables, beaucoup d'applications savent déjà manipuler le XML, et le RDF est un standard bien répandu. Le partage et l'échange dans ces formats en sont d'autant plus faciles.

Le Web sémantique peut donc profiter de ce format pour structurer, partager et échanger les différentes connaissances qui s'y trouvent. Il y a déjà plusieurs ontologies modélisées à l'aide d'OWL. »

http://fr.wikipedia.org/wiki/Web_Ontology_Language



1. évolution du web actuel au web sémantique par escalier en passant du RDF à l'owl

2.3. Memoria-Mea

Dans le contexte de PIM, différents partenaires de la HES-SO se sont regroupés autour d'un projet de recherche appliqué nommé Memoria Mea. L'un de ces partenaires la HES-SO Valais - dans laquelle se déroule ce travail de diplôme - se penche plus particulièrement sur les technologies de web sémantique

Le projet Memoria-Mea vise à développer un système sur un modèle sémantique permettant à une personne **d'organiser, de classifier et de rechercher tout type d'information multimédia numérique** avec laquelle elle interagi pendant ses activités quotidiennes (mails, documents, agenda, émissions TV ou radio, photos) http://www.memoria-mea.ch/

Considérant que les moteurs de recherche courants de multimédia sont conçus pour des grandes applications et le grand nombre d'utilisateur, la nouveauté du projet de **Memoria-Mea est de soutenir la mémoire individuelle**, c.-à-d. les environnements et les données personnels, en prolongeant des techniques existantes de recherche documentaire et de visualisation avec l'exploitation de données et les modèles ontologiques, basés sur le comportement d'utilisateur et de ses besoins

2.4. OntoMea

OntoMea est un prototype fournissant l'architecture pour manipuler la base de connaissance sémantique de Memoria-Mea. OntoMea est une base de connaissance locale qui fonctionne sur l'ordinateur local d'un utilisateur. Il manipule les données relatives à un utilisateur. Les utilisateurs ne sont pas reliés directement les uns aux autres mais ils pourraient s'échanger des données.

OntoMea est une application de Java 1.5 avec des web services inclus, celui n'exige aucune installation spéciale excepté Java 1.5 et OntoMea lui-même. Il a été examiné seulement sur la plateforme de Windows, mais des essais pourraient être faits sur d'autres plateformes au besoin.

OntoMea contient des données sémantiques provenant de différents modules de Memoria-Mea et fournit un moteur d'interrogation de ces données. Les informations sont représentées grâce à des ontologies

3. Problèmatique

Actuellement dans Memoria Mea, les informations sont en locales ce qui implique plusieurs restrictions telles que :

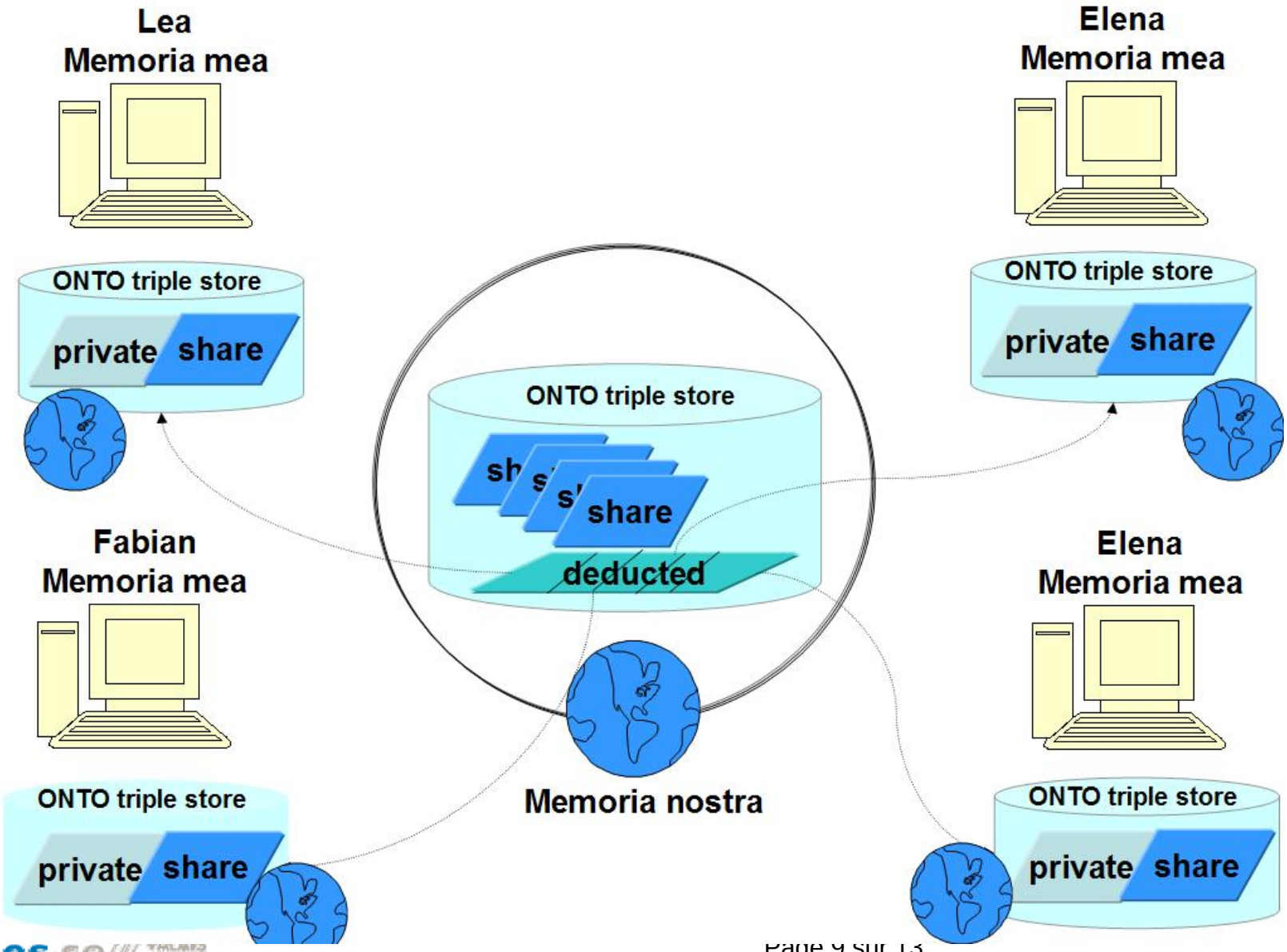
- un utilisateur ne peut pas y accéder lorsqu'il est sur une autre machine ou en voyage (intéressé de pouvoir déposer des informations nouvelles)
- Pas de réel partage de l'information entre utilisateur sinon par envoi de fichiers
- On pourrait profiter de la puissance des technologies du Web sémantique pour déduire de nouvelles informations lorsqu l'on met ensemble les données de plusieurs utilisateurs.

Le sujet de ce travail de diplôme est de créer une architecture et un démonstrateur qui passe d'un Memoria Mea local à un Memoria Nostra en réseau. (cf. graphique)

Mais ce passage comporte plusieurs points à prendre en considération :

- Etat de l'art : Analyse des différentes solutions qui pourraient aider à la réalisation du projet
- Appréhension de la problématique architecture globale
- Gestion des utilisateurs / réseau social
- Notion de partage de l'information (data)

Web 3.0 : Déploiement



4. Travail à effectuer

4.1. Etat de l'art

Analyse des différentes solutions qui pourraient aider à la réalisation du projet.

- Voir les solutions classiques non sémantiques existantes

Flickr est un site web gratuit (des fonctionnalités étendues sont payantes) de diffusion et de partage de photographies dont l'organisation s'apparente à celle d'une communauté virtuelle),

- Voir les solutions web sémantique existantes :

Piggy TheSemanticWeb (Piggy Bank est une extension propulsée par Java qui transforme Mozilla Firefox en un « navigateur du Web sémantique »).

Zitgist: un vrai moteur de recherche sémantique RDF

*« **Zitgist** est tout simplement un moteur de recherche sémantique. Certains se demandent ce qu'est un moteur de recherche sémantique ? Quelle différence avec un moteur de recherche traditionnel comme Google, Yahoo ! Ou MSN ? L'unique différence tient dans l'information recueillie, indexée et utilisée pour répondre aux requêtes des utilisateurs. Au lieu d'utiliser des documents lisibles par l'humain, comme le HTML, PDF ou DOC, Ztigist utilisera les documents sémantiques RDF. La caractéristique de ces documents est qu'ils décrivent des choses. En fait, ces documents peuvent décrire n'importe quoi. Et comme Zitgist comprend la signification de ces descriptions, les utilisateurs pourront poser de complexes requêtes et trouver tout ce qu'ils veulent ». Frédérick Giasson*

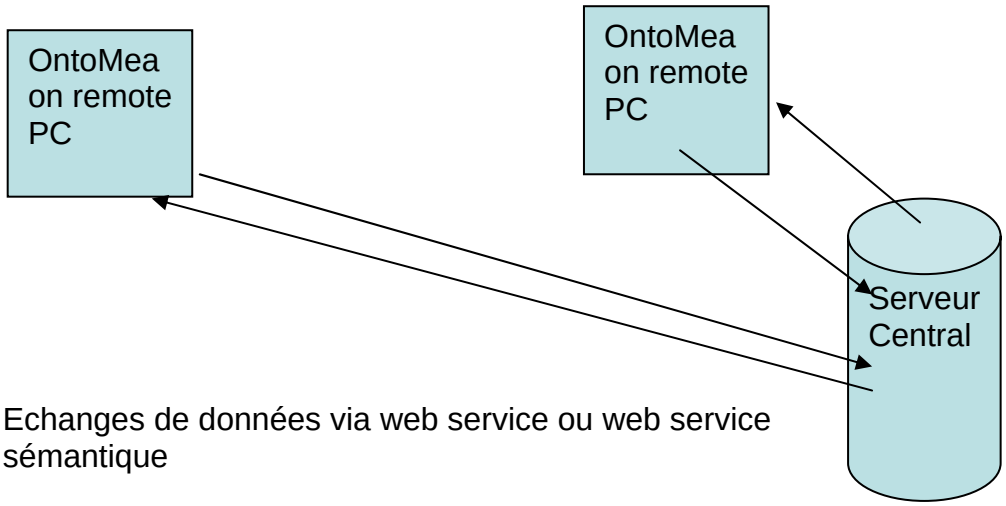
- **Délivrable** rapport d'analyse

4.2. Appréhension de la problématique architecture globale

L'objectif de ce travail de diplôme est de mettre en place une architecture permettant depuis la gestion de données locales, propres à un utilisateur, la mise en réseau des données de tous les utilisateurs du système sur un serveur central.

Il faudra tenir compte des problèmes liés à la **sécurité de** ces données, ainsi qu'à la **consistance et la mise à jour** entre les différentes localisations d'une même information.

.



4.3. Gestion des utilisateurs / réseau social

- Idée de 'carte d'identité' de personnes, qui permettraient ensuite de remplir automatiquement les différents formulaires du web (nom, prénom, etc..) en ne spécifiant qu'un URI pour aller rechercher l'info, et non pas remplir à chaque fois tous les champs. Exemple qui peut être utilisé http://www.facebook.com

FaceBook est un site Web de réseau social destiné à rassembler avec peut être des notions de partage de l'information privée/publique

- Recherche une solution pour l'URI (**Uniform Resource Identifier** qui identifie une ressource générique qui peut être une adresse Web, un document, une image, un service ou un URL)

4.4. Notion de partage de l'information

- Data publique / privée
- Transfert de l'information vers OntoNostra
- Retour d'information vers les différents OntoMea (quels infos, à qui, …)
- Utilisation du serveur central pour synchroniser différentes instances OntoMea d'une personne : ainsi des informations 'privées' peuvent transiter sur le serveur afin de mettre à jour d'autres moteurs locaux

5. Durée et déroulement

Le projet a un équivalent de 12 semaines à plein temps (20 crédits * 25-30 heures = 500-600 heures)

- 8h * 16 semaines = environ 130 heures durant le semestre hivernale
- 470h/ 24h = 20 semaines (16 semaines semestre été + 4 semaines inter semestre)

Début du projet :	**17 septembre 2007**
Remise du cahier des charger :	**15 octobre 2007**
Fin du projet :	**28 juin 2008**

6. Délivrable

6.1. Rapport d'analyse

Le but du rapport est d'apporter des réponses pour des questions comme le choix les outils adaptés et la faisabilité du projet tel qu'il est écrit actuellement. Ce rapport devra aider sur le choix du problème d'URI unique et du serveur central.

6.2. Proposition d'architecture

L'architecture actuellement prévue est la mise en réseau des données de tous les utilisateurs du système sur un serveur central en toute sécurité.

Cette proposition devra mettre en place une architecture permettant depuis la gestion de données locales, propres à un utilisateur, la mise en réseau des données de tous les utilisateurs du système sur un serveur central.

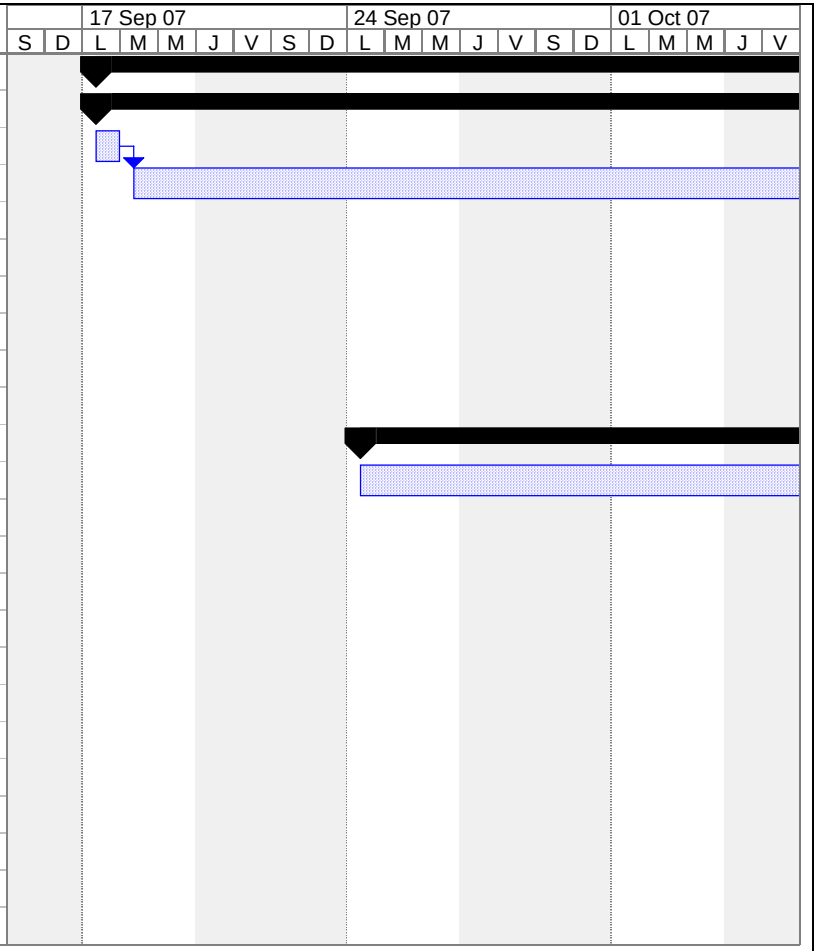
6.3. Mise en place d'un démonstrateur (Serveur central)

La solution devra prendre en compte la possibilité de mettre des informations à jour depuis n'importe quel accès Internet dans le monde (par exemple lors d'un voyage). Tout en assurant la validité et la cohérence entre les données centralisées et les données locales.

7. Conclusion

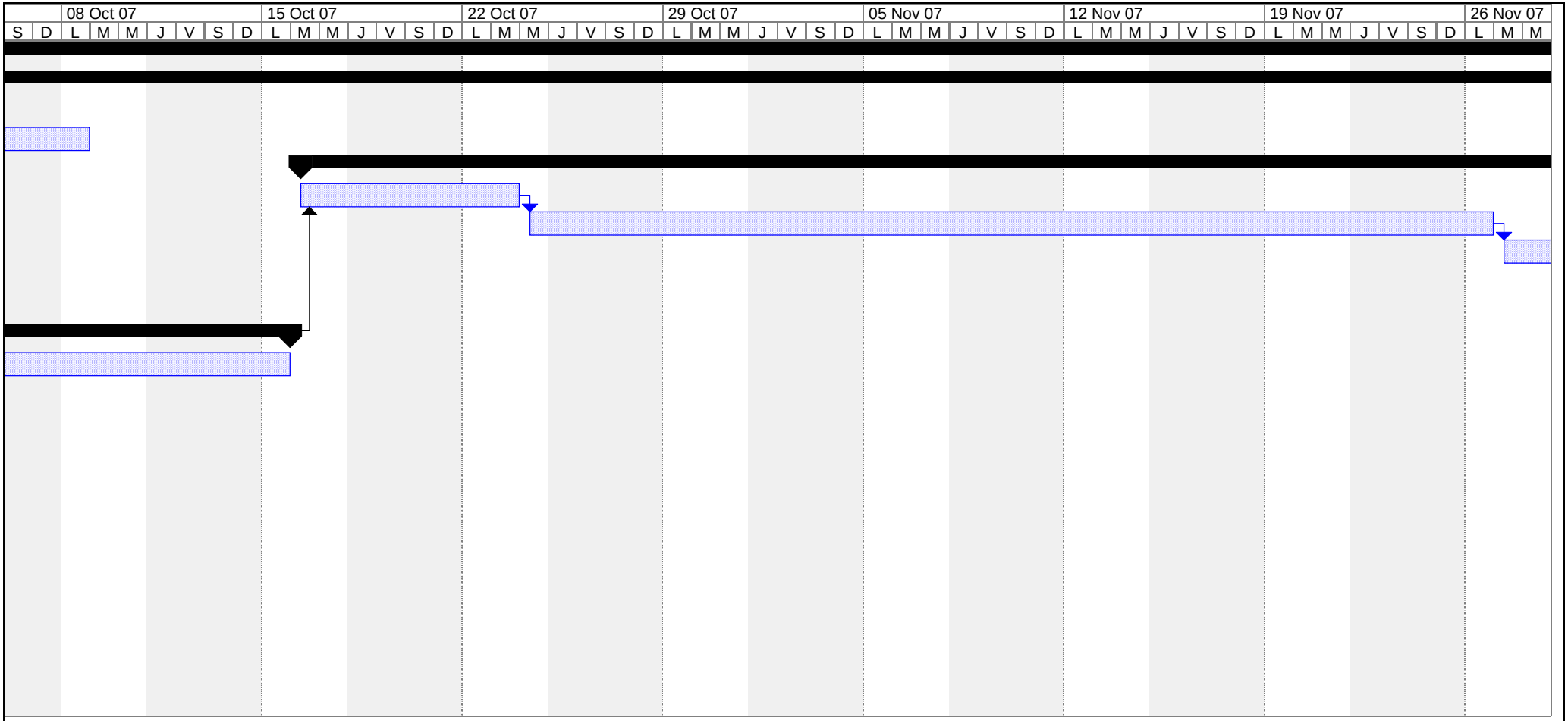
Ce cahier des charges cherche à montrer l'état des faits actuels mais qui pourrait changer grâce au rapport d'analyse. La plus grande problématique est que le web sémantique étant un jeune outil informatique il évolue très rapidement. Ce qui est juste actuellement pourrait changer complètement. L'appréhension de nouveaux langages tels qu'OWL, RDF ou SPARQL pourrait poser des problèmes, mais sera très stimulant.

N°	Nom de la tâche	Durée	Début	Fin
1	**Projet Final**	**111 jours**	**Lun 17.09.07**	**Sam 28.06.08**
2	**Initialisation**	**38 jours**	**Lun 17.09.07**	**Mar 11.12.07**
3	Lancement de la taches	1 jour	Lun 17.09.07	Lun 17.09.07
4	Appréhension du web sémantique	9 jours	Mar 18.09.07	Lun 08.10.07
5	**Etude de faisabilité**	**25 jours**	**Mar 16.10.07**	**Mar 11.12.07**
6	Etablir les limites du projets	4 jours	Mar 16.10.07	Mar 23.10.07
7	Etablir les fonctionnalités à l'aide des respon	14 jours	Mer 24.10.07	Lun 26.11.07
8	Définir les outils techniques (Web? Win32 , S	4 jours	Mar 27.11.07	Mar 04.12.07
9	Etablir une gestion des risques (liste et suivi)	2 jours	Mer 05.12.07	Lun 10.12.07
10	Validation de l'étape avec la direction (cahier	1 jour	Mar 11.12.07	Mar 11.12.07
11	**Cadrage**	**10 jours**	**Lun 24.09.07**	**Lun 15.10.07**
12	Ecriture du cahier des charges	10 jours	Lun 24.09.07	Lun 15.10.07
13	**Design**	**13 jours**	**Mer 12.12.07**	**Mer 06.02.08**
14	Réaliser le modèle conceptuel de données	5 jours	Mer 12.12.07	Lun 21.01.08
15	Réaliser les écrans les utilisateurs	7 jours	Mar 22.01.08	Mar 05.02.08
16	Validation de l'étape avec les utilisateurs et la dire	1 jour	Mer 06.02.08	Mer 06.02.08
17	**Construction**	**38 jours**	**Lun 11.02.08**	**Mar 06.05.08**
18	Développement du logiciel	32 jours	Lun 11.02.08	Mar 22.04.08
19	Documentation technique	6 jours	Mer 23.04.08	Mar 06.05.08
20	**Finalisation**	**22 jours**	**Mer 07.05.08**	**Sam 28.06.08**
21	Etablissement du prototype final	6 jours	Mer 07.05.08	Mar 20.05.08
22	Validation par la direction-utilisateurs	3 jours	Mer 21.05.08	Mar 27.05.08
23	Déploiement de l'application	3 jours	Mer 28.05.08	Mar 03.06.08
24	Clôture du projet	0 jour	Sam 28.06.08	Sam 28.06.08



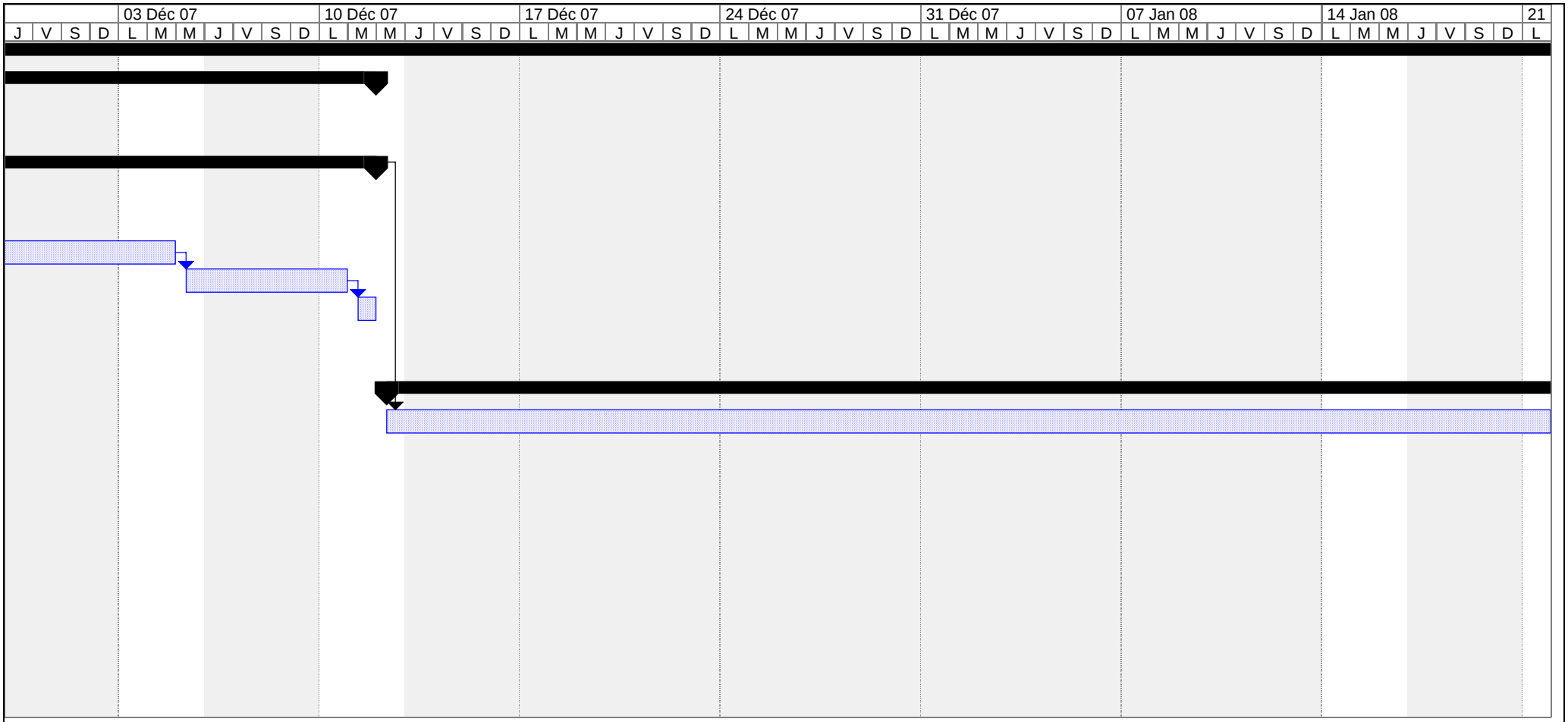
Projet : Projet1
Date : Ven 20.06.08

Tâche
Fractionnement
Avancement
Jalon
Récapitulative
Récapitulatif du projet
Fractionnement
Avancement
Jalon



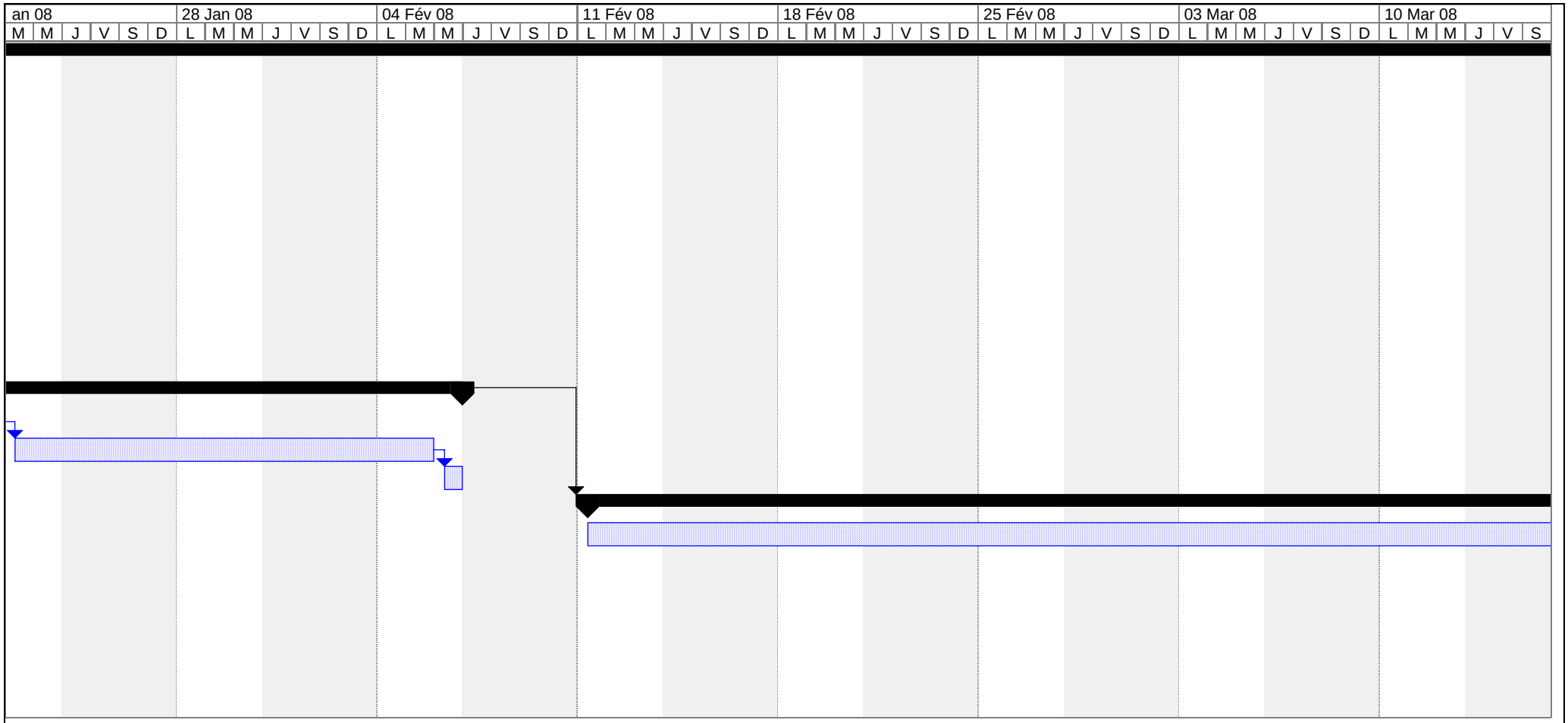
Projet : Projet1
Date : Ven 20.06.08

Tâche		Jalon		Fractionnement	
Fractionnement		Récapitulative		Avancement	
Avancement		Récapitulatif du projet		Jalon	



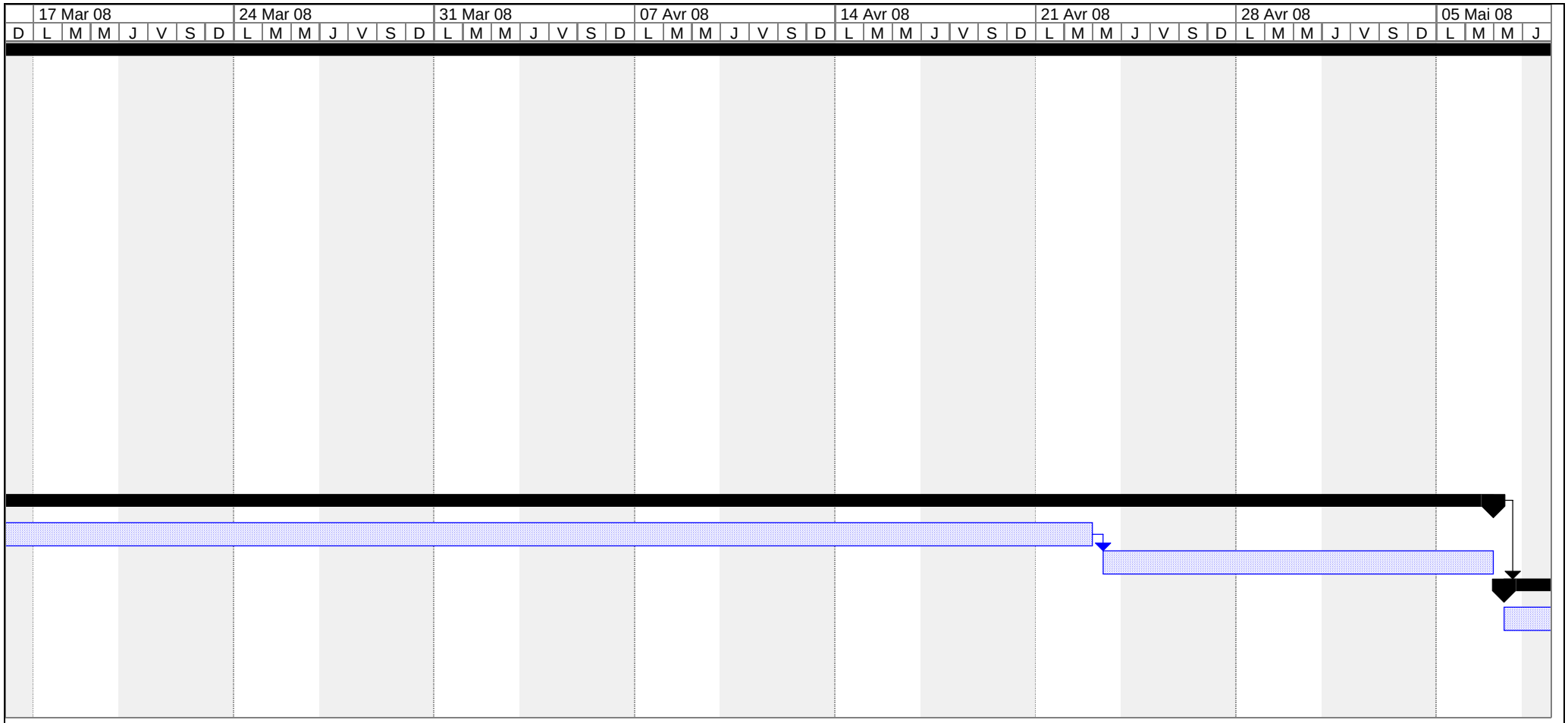
Projet : Projet1
Date : Ven 20.06.08

Tâche		Jalon		Fractionnement	
Fractionnement		Récapitulative		Avancement	
Avancement		Récapitulatif du projet		Jalon	



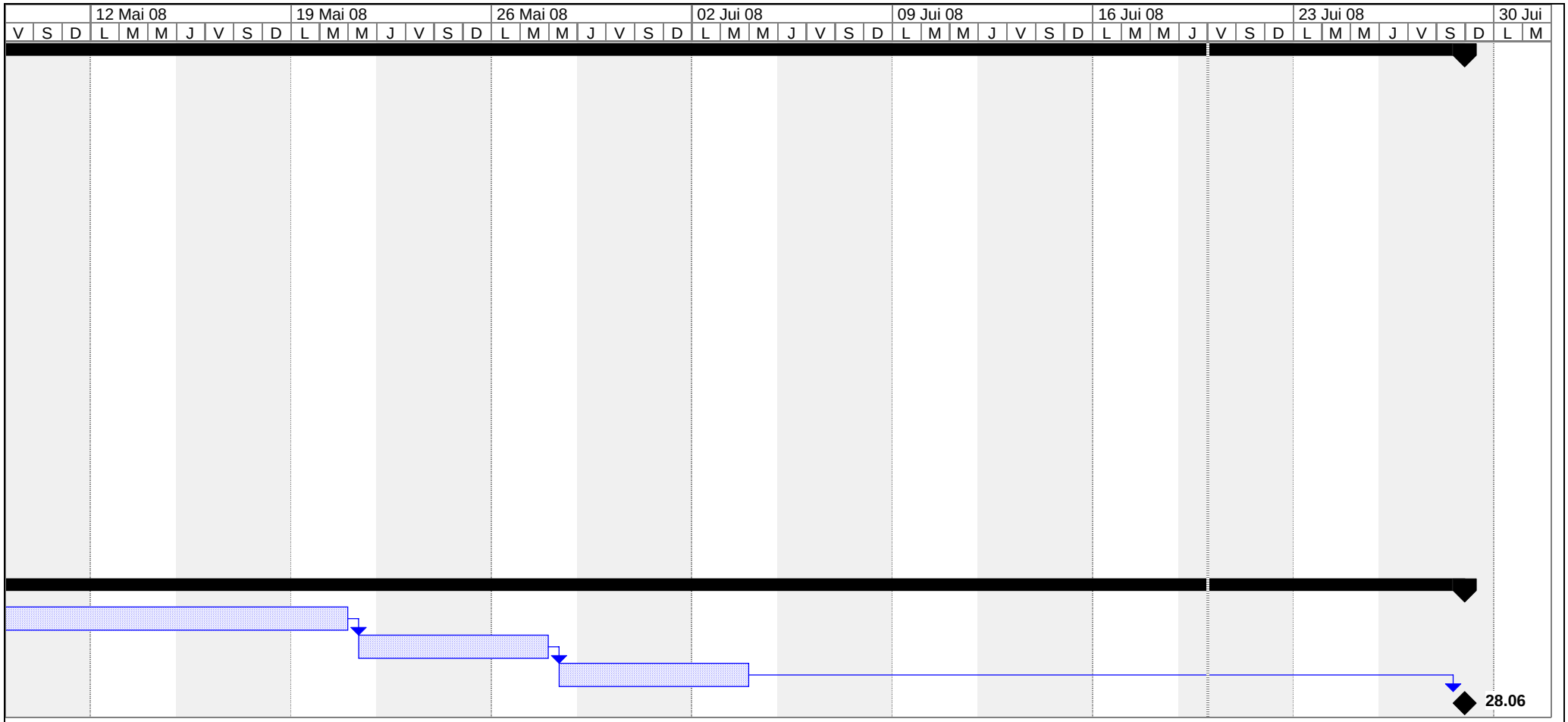
Projet : Projet1
Date : Ven 20.06.08

Tâche		Jalon		Fractionnement	
Fractionnement		Récapitulative		Avancement	
Avancement		Récapitulatif du projet		Jalon	



Projet : Projet1
Date : Ven 20.06.08

Tâche		Jalon		Fractionnement	
Fractionnement		Récapitulative		Avancement	
Avancement		Récapitulatif du projet		Jalon	



Projet : Projet1
Date : Ven 20.06.08

Tâche		Jalon		Fractionnement	
Fractionnement		Récapitulative		Avancement	
Avancement		Récapitulatif du projet		Jalon	

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

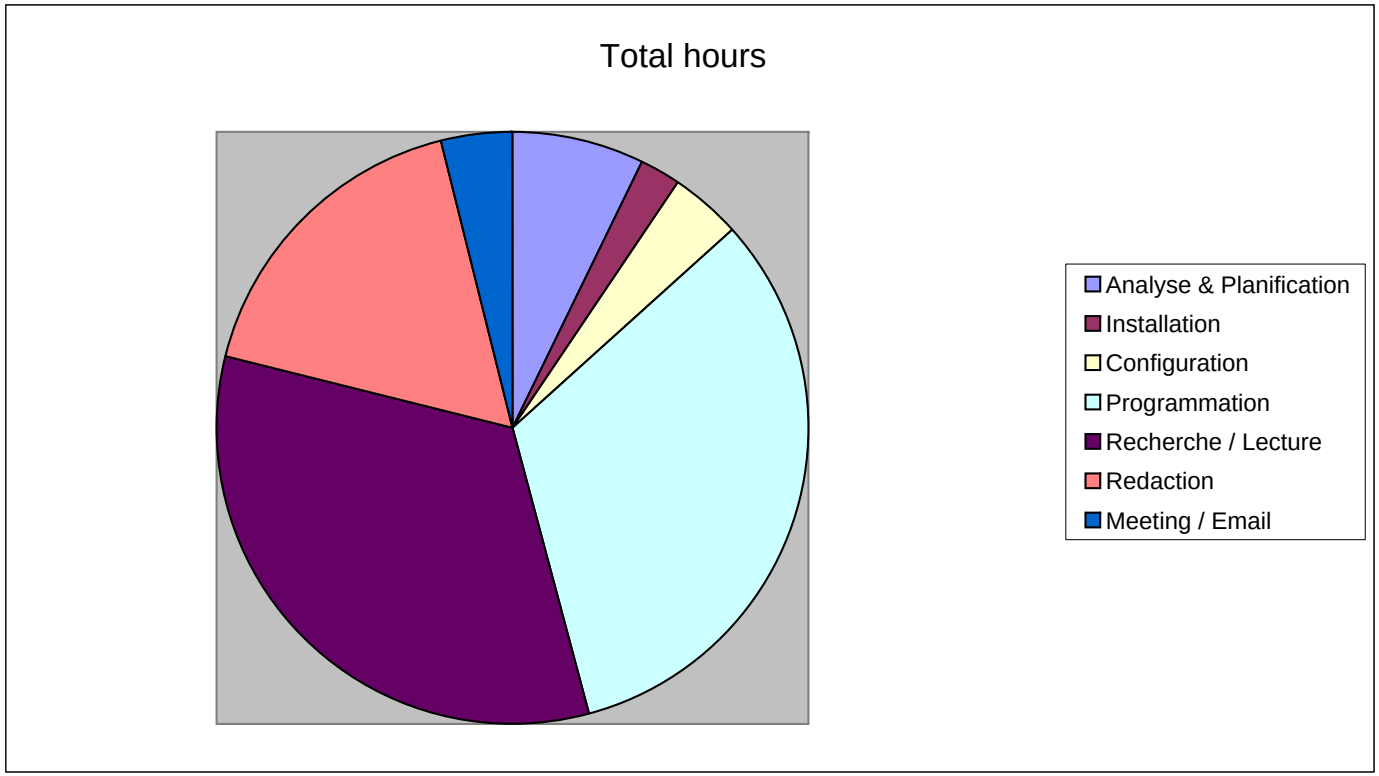
HEVs **Report Summary**

Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Analyse & Planification	0.00	1.00	5.00	3.00	2.00	3.00	2.00	3.00	0.00	1.00	1.00	0.00	3.00	0.00	6.00	4.00	1.00	1.00	1.00
Installation	0.00	0.00	2.00	3.00	1.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	3.00
Configuration	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	4.00	3.00	0.00	2.00	1.00
Programmation	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	5.00	5.00	11.00	8.00
Recherche / Lecture	2.00	23.00	9.00	13.00	15.00	11.00	5.00	8.00	2.00	3.00	6.00	6.00	5.00	4.00	4.00	3.00	2.00	4.00	4.00
Redaction	0.00	0.00	5.00	5.00	2.00	2.00	1.00	0.00	0.00	2.00	0.00	4.00	3.00	3.00	2.00	2.00	0.00	0.00	0.00
Meeting / Email	2.00	0.00	2.00	0.00	0.00	3.00	0.00	1.00	1.00	0.00	1.00	2.00	0.00	0.00	0.00	1.00	1.00	0.00	1.00
Total	**4.00**	**24.00**	**24.00**	**24.00**	**20.00**	**20.00**	**8.00**	**12.00**	**3.00**	**7.00**	**8.00**	**12.00**	**11.00**	**10.00**	**17.00**	**18.00**	**11.00**	**19.00**	**18.00**

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	Total hours
0.00	1.00	3.00	2.00	0.00	1.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00	2.00	2.00	2.00	0.50	0.00	0.00	0.00	52.50
0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	16.00
1.00	0.00	0.00	1.00	0.00	0.00	2.00	1.00	1.00	2.00	2.00	4.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	28.00
8.00	10.00	10.00	8.00	1.00	9.00	17.00	11.00	10.00	10.00	14.00	6.00	13.00	3.50	12.00	7.00	1.00	15.00	16.00	19.00	232.50
8.00	2.00	3.00	3.00	1.00	2.00	4.00	5.00	3.00	6.00	7.00	6.00	18.00	10.00	12.00	7.00	6.00	2.00	3.00	0.00	237.00
2.00	6.00	3.00	0.00	0.00	0.00	0.00	1.00	1.00	0.00	0.00	0.00	0.00	7.00	6.50	11.00	19.00	14.00	13.00	9.00	123.50
2.00	0.00	0.00	2.00	0.00	1.00	0.00	1.00	2.00	1.00	0.00	0.50	0.00	1.50	0.00	0.50	0.50	0.00	0.00	0.00	27.00
21.00	19.00	19.00	17.00	2.00	13.00	23.00	20.00	17.00	19.00	24.00	16.50	34.00	24.00	32.50	27.50	27.00	31.00	32.00	28.00	716.50

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Title	hours
Analyse & Planification	52.50
Installation	16.00
Configuration	28.00
Programmation	232.50
Recherche / Lecture	237.00
Redaction	123.50
Meeting / Email	27.00
Total	716.50



Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	09/17/2007-09/23/2007							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification								0.00
Installation								0.00
Configuration								0.00
Programmation								0.00
Research / Lecture					**2.00**			**2.00**
Redaction								0.00
Meeting				**2.00**				**2.00**
Total	0.00	0.00	0.00	**2.00**	**2.00**	0.00	0.00	**4.00**

Working Report Details

Date	Note
	Lancement du projet

Date Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	09/24/2007-09/30/2007							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification	**1.00**							**1.00**
Installation								0.00
Configuration								0.00
Programmation								0.00
Research / Lecture	**5.00**	**7.00**	**7.00**	**4.00**				**23.00**
Redaction								0.00
Meeting								0.00
Total	**6.00**	**7.00**	**7.00**	**4.00**	0.00	0.00	0.00	**24.00**

Working Report Details

Date	Note

Date Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	10/1/2007-10/8/2007	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification	2.00	2.00	1.00					5.00
Installation		1.00	1.00					2.00
Configuration		1.00						1.00
Programmation								0.00
Research / Lecture	3.00		4.00	2.00				9.00
Redaction		3.00		2.00				5.00
Meeting	2.00							2.00
Total	7.00	7.00	6.00	4.00	0.00	0.00	0.00	24.00

Working Report Details

Date	Note
	Réunion avec responsable de 15h à 16h40

Date Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	10/8/2007-10/14/2007	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification	3.00							3.00
Installation			1.00	2.00				3.00
Configuration								0.00
Programmation								0.00
Research / Lecture	3.00	5.00	4.00	1.00				13.00
Redaction	1.00	2.00	2.00					5.00
Meeting								0.00
Total	7.00	7.00	7.00	3.00	0.00	0.00	0.00	24.00

Working Report Details

Date	Note

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	10/15/2007-10/21/2007							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification		**2.00**						**2.00**
Installation	**1.00**							**1.00**
Configuration								0.00
Programmation								0.00
Research / Lecture	**5.00**	**3.00**	**5.00**	**2.00**				**15.00**
Redaction	**1.00**	**1.00**						**2.00**
Meeting								0.00
Total	**7.00**	**6.00**	**5.00**	**2.00**	0.00	0.00	0.00	**20.00**

Working Report Details

Date	Note

Date Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	10/22/2007-10/28/2007							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification	**2.00**		**1.00**					**3.00**
Installation								0.00
Configuration	**1.00**							**1.00**
Programmation								0.00
Research / Lecture	**2.00**	**5.00**	**4.00**					**11.00**
Redaction			**2.00**					**2.00**
Meeting	**2.00**		**1.00**					**3.00**
Total	**7.00**	**5.00**	**8.00**	0.00	0.00	0.00	0.00	**20.00**

Working Report Details

Date	Note
22.10.2007	Réunion avec Anne et Fabian

Date	Signature Professor
Comment	

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	10/29/2007-11/04/2007	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification	**2.00**							**2.00**
Installation								0.00
Configuration								0.00
Programmation								0.00
Research / Lecture	**3.00**	**2.00**						**5.00**
Redaction	**1.00**							**1.00**
Meeting								0.00
Total	**6.00**	**2.00**	0.00	0.00	0.00	0.00	0.00	**8.00**

Working Report Details

Date	Note

Date Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

FFHS	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	11/05/2007-12/11/2007	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification				1.00		2.00		3.00
Installation								0.00
Configuration								0.00
Programmation								0.00
Research / Lecture			1.00	2.00	1.00	4.00		8.00
Redaction								0.00
Meeting				1.00				1.00
Total	0.00	0.00	1.00	4.00	1.00	6.00	0.00	12.00

Working Report Details

Date	Note
08.11.2007	Rencontre avec fabian et anne frederic favre présentation ontomea 14h-14h45

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	19/11/2007 - 25/11/2007	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0.00
Installation								0.00
Configuration								0.00
Programmation								0.00
Research / Lecture						**2.00**		**2.00**
Redaction								0.00
Meeting					**1.00**			**1.00**
Total	0.00	0.00	0.00	0.00	**1.00**	**2.00**	0.00	**3.00**

Working Report Details

Date	Note

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

FFHS	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	26/11/2007 - 02/12/2007							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification						**1.00**		**1.00**
Installation							**1.00**	**1.00**
Configuration								0.00
Programmation								0.00
Research / Lecture						**2.00**	**1.00**	**3.00**
Redaction						**2.00**		**2.00**
Meeting								0.00
Total	0.00	0.00	0.00	0.00	0.00	**5.00**	**2.00**	**7.00**

Working Report Details	
Date	Note

Date	Signature Professor
Comment	

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	03/12/2007 - 09/12/2007	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification				1.00				1.00
Installation								0.00
Configuration								0.00
Programmation								0.00
Research / Lecture		1.00	1.00		1.00		3.00	6.00
Redaction								0.00
Meeting		1.00						1.00
Total	0.00	2.00	1.00	1.00	1.00	0.00	3.00	8.00

Working Report Details

Date	Note

Date Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	10/12/2007 - 16/12/2007	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0.00
Installation								0.00
Configuration								0.00
Programmation								0.00
Research / Lecture					**1.00**	**4.00**	**1.00**	**6.00**
Redaction							**4.00**	**4.00**
Meeting					**2.00**			**2.00**
Total	0.00	0.00	0.00	0.00	**3.00**	**4.00**	**5.00**	**12.00**

Working Report Details

Date	Note
14.12.2007	Rencontre avec Fabian et Anne parler de refaire l'etat de l'art plus proche de la réalité faciliter le language et recu travail diplôme de Frédéric
15.12.2007	Lecture du travail de diplôme
16.12.2007	Rédaction de l'etat de l'art

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	17/12/2007 - 23/12/2007							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification					**3.00**			**3.00**
Installation								0.00
Configuration								0.00
Programmation								0.00
Research / Lecture					**3.00**	**2.00**		**5.00**
Redaction						**3.00**		**3.00**
Meeting								0.00
Total	0.00	0.00	0.00	0.00	**6.00**	**5.00**	0.00	**11.00**

Working Report Details	
Date	Note

Date	Signature Professor
Comment	

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	24/12/2007 - 30/12/2007							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification								0.00
Installation								0.00
Configuration						**1.00**		**1.00**
Programmation					**1.00**	**1.00**		**2.00**
Research / Lecture			**2.00**	**2.00**				**4.00**
Redaction						**3.00**		**3.00**
Meeting								0.00
Total	0.00	0.00	**2.00**	**2.00**	**1.00**	**5.00**	0.00	**10.00**

Working Report Details	
Date	Note

Date	Signature Professor
Comment	

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	31/12/2007 - 06/01/2008							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification			**1.00**		**3.00**	**1.00**	**1.00**	**6.00**
Installation								0.00
Configuration						**4.00**		**4.00**
Programmation					**1.00**			**1.00**
Research / Lecture			**3.00**	**1.00**				**4.00**
Redaction			**1.00**				**1.00**	**2.00**
Meeting								0.00
Total	0.00	0.00	**5.00**	**1.00**	**4.00**	**5.00**	**2.00**	**17.00**

Working Report Details	
Date	Note

Date	Signature Professor
Comment	

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	07/01/2008 - 13/01/2008							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification				**2.00**		**1.00**	**1.00**	**4.00**
Installation								0.00
Configuration				**1.00**	**2.00**			**3.00**
Programmation					**2.00**	**3.00**		**5.00**
Research / Lecture				**2.00**			**1.00**	**3.00**
Redaction				**1.00**			**1.00**	**2.00**
Meeting					**1.00**			**1.00**
Total	0.00	0.00	0.00	**6.00**	**5.00**	**4.00**	**3.00**	**18.00**

Working Report Details	
Date	Note

Date	Signature Professor
Comment	

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	14/1/2008-20/1/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification				1.00				1.00
Installation					1.00	1.00		2.00
Configuration								0.00
Programmation				1.00		4.00		5.00
Research / Lecture					2.00			2.00
Redaction								0.00
Meeting	1.00							1.00
Total	1.00	0.00	0.00	2.00	3.00	5.00	0.00	11.00

Working Report Details

Date	Note

Date Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	21/1/2008-27/1/2008							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification					**1.00**			**1.00**
Installation				**1.00**				**1.00**
Configuration				**1.00**	**1.00**			**2.00**
Programmation				**2.00**	**4.00**	**4.00**	**1.00**	**11.00**
Research / Lecture					**2.00**		**2.00**	**4.00**
Redaction								0.00
Meeting								0.00
Total	0.00	0.00	0.00	**4.00**	**8.00**	**4.00**	**3.00**	**19.00**

Working Report Details

Date	Note

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	01/28/2008-03/02/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification					1.00			1.00
Installation					2.00	1.00		3.00
Configuration						1.00		1.00
Programmation				2.00	4.00	2.00		8.00
Research / Lecture				2.00	1.00	1.00		4.00
Redaction								0.00
Meeting	1.00							1.00
Total	1.00	0.00	0.00	4.00	8.00	5.00	0.00	18.00

Working Report Details

Date	Note

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	04/02/2008-11/02/2008							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification								0.00
Installation								0.00
Configuration						**1.00**		**1.00**
Programmation				**1.00**	**1.00**	**3.00**	**3.00**	**8.00**
Research / Lecture	**0.50**	**0.50**	**1.00**	**2.00**	**1.00**	**1.00**	**2.00**	**8.00**
Redaction					**2.00**			**2.00**
Meeting					**2.00**			**2.00**
Total	**0.50**	**0.50**	**1.00**	**3.00**	**6.00**	**5.00**	**5.00**	**21.00**

Working Report Details

Date	Note
08.02.2008	Rencontre avec anne le calve et Fabian

Date	Signature Professor
Comment	

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

FFHS	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	02/11/2008-02/18/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification		1.00						1.00
Installation								0.00
Configuration								0.00
Programmation	1.00	1.00			7.50		0.50	10.00
Research / Lecture	1.00					1.00		2.00
Redaction			1.00	1.00			4.00	6.00
Meeting								0.00
Total	2.00	2.00	1.00	1.00	7.50	1.00	4.50	19.00

Working Report Details

Date	Note
15.02.2008	Configuration Openid et appréhension d'easy php et apache
17.02.2008	Ecriture de l'état d'art

Date	Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	18/02/2008 - 24/02/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification					1.00	2.00		3.00
Installation								0.00
Configuration								0.00
Programmation		1.00	1.00		4.00		4.00	10.00
Research / Lecture		1.00			2.00			3.00
Redaction	1.00				1.00	1.00		3.00
Meeting								0.00
Total	1.00	2.00	1.00	0.00	8.00	3.00	4.00	19.00

Working Report Details

Date	Note
22.02.2008	Jena et web service tomcat
23.02.2008	Choix des fonctions web service

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

FFHS	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	25/02/2008 - 02/03/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification					2.00			2.00
Installation						1.00		1.00
Configuration						1.00		1.00
Programmation	1.00	1.00			1.00	1.00	4.00	8.00
Research / Lecture			1.00		2.00			3.00
Redaction								0.00
Meeting		0.50	0.50		1.00			2.00
Total	1.00	1.50	1.50	0.00	6.00	3.00	4.00	17.00

Working Report Details

Date	Note
29.02.2008	Rencontre avec F. Cretton sur les differente chose encre a faire

Date Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	03/03/2008 - 08/03/2008							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification								0.00
Installation								0.00
Configuration								0.00
Programmation		**1.00**						**1.00**
Research / Lecture	**1.00**							**1.00**
Redaction								0.00
Meeting								0.00
Total	**1.00**	**1.00**	0.00	0.00	0.00	0.00	0.00	**2.00**

Working Report Details

Date	Note

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	10/03/2008 - 16/03/2008							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification					**1.00**			**1.00**
Installation								0.00
Configuration								0.00
Programmation					**4.00**	**2.00**	**3.00**	**9.00**
Research / Lecture					**2.00**			**2.00**
Redaction								0.00
Meeting					**1.00**			**1.00**
Total	0.00	0.00	0.00	0.00	**8.00**	**2.00**	**3.00**	**13.00**

Working Report Details

Date	Note
14.03.2008	Rencontre avec Fabian sur comment doivent se parler deux ontomea lointain

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	17/03/2007 - 23/03/2007							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification								0.00
Installation								0.00
Configuration					**1.00**		**1.00**	**2.00**
Programmation				**7.00**	**5.00**	**4.00**	**1.00**	**17.00**
Research / Lecture				**2.00**	**2.00**			**4.00**
Redaction								0.00
Meeting								0.00
Total	0.00	0.00	0.00	**9.00**	**8.00**	**4.00**	**2.00**	**23.00**

Working Report Details	
Date	Note
	certificats apache ou iis
	Web Service c#.net et Axiis

Date	Signature Professor
Comment	

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	24/03/2008 - 30/03/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification							1.00	1.00
Installation								0.00
Configuration						1.00		1.00
Programmation		1.00			5.00	4.00	1.00	11.00
Research / Lecture					1.00	2.00	2.00	5.00
Redaction	1.00							1.00
Meeting					1.00			1.00
Total	1.00	1.00	0.00	0.00	7.00	7.00	4.00	20.00

Working Report Details

Date	Note
28.03.2008	Rencontre avec Fabian
	Certificat sur apache et web service

Date	Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	31/04/2008 - 06/01/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0.00
Installation								0.00
Configuration							1.00	1.00
Programmation		1.00		1.00	5.00	1.00	2.00	10.00
Research / Lecture						1.00	2.00	3.00
Redaction						1.00		1.00
Meeting	1.00				1.00			2.00
Total	1.00	1.00	0.00	1.00	6.00	3.00	5.00	17.00

Working Report Details

Date	Note
04.04.2008	Rencontre avec fabian

Date	Signature Professor
Comment	

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report							
Title	Web 3.0 - OntoNostra							
Name	DePalma Francesco Nicola							
Week	07/04/2008 - 13/04/2008							
	Mo	Tu	We	Th	Fr	Sa	So	**Total**
Analysis & Planification								0.00
Installation								0.00
Configuration					**2.00**			**2.00**
Programmation					**4.00**	**3.00**	**3.00**	**10.00**
Research / Lecture	**1.00**				**2.00**	**1.00**	**2.00**	**6.00**
Redaction								0.00
Meeting	**0.50**	**0.50**						**1.00**
Total	**1.50**	**0.50**	0.00	0.00	**8.00**	**4.00**	**5.00**	**19.00**

Working Report Details

Date	Note
	Certificat + test de montée en charge d'un fichier rdf de 4GO

Date	Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

FFHS	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	14/07/2008-20/07/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0.00
Installation							1.00	1.00
Configuration					1.00	1.00		2.00
Programmation		1.00	0.50	1.50	6.00	2.00	3.00	14.00
Research / Lecture	1.00		0.50	0.50	1.00	2.00	2.00	7.00
Redaction								0.00
Meeting								0.00
Total	1.00	1.00	1.00	2.00	8.00	5.00	6.00	24.00

Working Report Details

Date	Note

Date	Signature Professor
Comment	

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	21/04/2008 - 27/04/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0.00
Installation								0.00
Configuration						4.00		4.00
Programmation			1.00			3.00	2.00	6.00
Research / Lecture	1.00	1.00				2.00	2.00	6.00
Redaction								0.00
Meeting				0.50				0.50
Total	**1.00**	**1.00**	**1.00**	**0.50**	0.00	**9.00**	**4.00**	**16.50**

Working Report Details

Date	Note
	Axis2 et web service

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

FFHS	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	28/04/2007 - 04/05/2007	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification					1.00			1.00
Installation						1.00		1.00
Configuration					1.00			1.00
Programmation	2.00		1.00	5.00	3.00		2.00	13.00
Research / Lecture	1.00	1.00		7.00	3.00	3.00	3.00	18.00
Redaction								0.00
Meeting								0.00
Total	3.00	1.00	1.00	12.00	8.00	4.00	5.00	34.00

Working Report Details

Date	Note
	Recherche MTOM et Security avec JAXWS-ri

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	05/05/2008 - 11/05/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification					2.00			2.00
Installation								0.00
Configuration								0.00
Programmation	1.00			1.00	1.50			3.50
Research / Lecture	1.00	2.00		1.00	4.00	1.00	1.00	10.00
Redaction			2.00			2.00	3.00	7.00
Meeting	0.50				1.00			1.50
Total	2.50	2.00	2.00	2.00	8.50	3.00	4.00	24.00

Working Report Details

Date	Note
09.05.2008	Rencontre avec Fabian et Anne pour faire une mise au point du projet

Date	Signature Professor

Comment

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	12/05/2008 - 18/05/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification	1.00				1.00			2.00
Installation								0.00
Configuration								0.00
Programmation					5.00	4.00	3.00	12.00
Research / Lecture	1.00	1.00	1.00	1.00	4.00	2.00	2.00	12.00
Redaction	2.00	1.00	1.50	1.00			1.00	6.50
Meeting								0.00
Total	**4.00**	**2.00**	**2.50**	**2.00**	**10.00**	**6.00**	**6.00**	**32.50**

Working Report Details

Date	Note
	Web Service Mtom + installation Wse 3.0

Date	Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	19/05/2008 - 26/05/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification					1.00		1.00	2.00
Installation								0.00
Configuration								0.00
Programmation		2.00	2.00	3.00				7.00
Research / Lecture	2.00		1.00	2.00	1.00		1.00	7.00
Redaction				1.00	3.00	3.00	4.00	11.00
Meeting		0.50						0.50
Total	2.00	2.50	3.00	6.00	5.00	3.00	6.00	27.50

Working Report Details

Date	Note
	Ecriture du document

Date	Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	27/05/2008 - 01/06/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification					0.50			0.50
Installation								0.00
Configuration								0.00
Programmation					1.00			1.00
Research / Lecture		1.00	1.00		2.00	1.00	1.00	6.00
Redaction	1.00		1.00	2.00	6.00	5.00	4.00	19.00
Meeting					0.50			0.50
Total	1.00	1.00	2.00	2.00	10.00	6.00	5.00	27.00

Working Report Details

Date	Note

Date Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	02/06/2008 - 08/06/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0.00
Installation								0.00
Configuration								0.00
Programmation	1.00	2.00			6.00	3.00	3.00	15.00
Research / Lecture					1.00		1.00	2.00
Redaction	1.00		2.00	3.00	1.00	4.00	3.00	14.00
Meeting								0.00
Total	2.00	2.00	2.00	3.00	8.00	7.00	7.00	31.00

Working Report Details

Date	Note
	Web Services + documentation

Date	Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	09/06/2008 - 15/06/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0.00
Installation								0.00
Configuration								0.00
Programmation		1.00		2.00	4.00	5.00	4.00	16.00
Research / Lecture			1.00			2.00		3.00
Redaction	1.00	3.00	2.00	2.00	1.00	2.00	2.00	13.00
Meeting								0.00
Total	1.00	4.00	3.00	4.00	5.00	9.00	6.00	32.00

Working Report Details

Date	Note
	Web Services + documentation

Date	Signature Professor

Comment

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HEVs	Weekly Report	
Title	Web 3.0 - OntoNostra	
Name	DePalma Francesco Nicola	
Week	16/06/2008 - 22/06/2008	

	Mo	Tu	We	Th	Fr	Sa	So	Total
Analysis & Planification								0.00
Installation								0.00
Configuration								0.00
Programmation	1.00	1.00	1.00		6.00	6.00	4.00	19.00
Research / Lecture								0.00
Redaction	2.00	1.00	1.00	2.00	1.00		2.00	9.00
Meeting								0.00
Total	3.00	2.00	2.00	2.00	7.00	6.00	6.00	28.00

Working Report Details

Date	Note
	Web Services + documentation + test

Date Signature Professor

Comment

Cool URIs for the Semantic Web

W3C Interest Group Note 31 March 2008

This version:
http://www.w3.org/TR/2008/NOTE-cooluris-20080331/
Latest version:
http://www.w3.org/TR/cooluris/
Previous version:
http://www.w3.org/TR/2008/WD-cooluris-20080321/
Editors:
Leo Sauermann (DFKI GmbH)
Richard Cyganiak (DERI, NUI Galway and Freie Universität Berlin)
Contributors:
Danny Ayers (Talis Information Ltd.)
Max Völkel (FZI Karlsruhe)

Abstract

The *Resource Description Framework* RDF allows users to describe both Web documents and concepts from the real world—people, organisations, topics, things—in a computer-processable way. Publishing such descriptions on the Web creates the *Semantic Web*. URIs (Uniform Resource Identifiers) are very important, providing both the core of the framework itself and the link between RDF and the Web. This document presents guidelines for their effective use. It discusses two strategies, called *303 URIs* and *hash URIs*. It gives pointers to several Web sites that use these solutions, and briefly discusses why several other proposals have problems.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is a W3C Interest Group Note giving a tutorial explaining decisions of the TAG for newcomers to Semantic Web technologies. It was initially based on the DFKI Technical Memo TM-07-01, *Cool URIs for the Semantic Web* and was subsequently published as a W3C Working draft in December 2007, and again in March 2008 by the Semantic Web Education and Outreach (SWEO) Interest Group of the W3C, part of the W3C Semantic Web Activity. The drafts were publicly reviewed, especially by the Technical Architecture Group (TAG) and the Semantic Web Deployment Group (SWD).

The charter of the Semantic Web Education and Outreach (SWEO) Interest Group expired at the end of March, 2008. Nevertheless, this document may be taken up by some other groups in the future for further development. Feedbacks on this documents is therefore encouraged. Please send comments about this document to public-sweo-ig@w3.org (with public archive). A complete list of changes is available.

Publication as an Interest Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. The

group does not expect this document to become a W3C Recommendation. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

The disclosure obligations of the Participants of this group are described in the charter.

Scope

This document is a practical guide for implementers of the RDF specification. The title is inspired by Tim Berners-Lee's article "Cool URIs don't change" [Cool]. It explains two approaches for RDF data hosted on HTTP servers. Intended audiences are Web and ontology developers who have to decide how to model their RDF URIs for use with HTTP. Applications using non-HTTP URIs are not covered. This document is an informative guide covering selected aspects of previously published, detailed technical specifications. The 303 URIs are based on the httpRange-14 resolution [*httpRange*] by the Technical Architecture Group (TAG). We assume that you are familiar with the basics of the RDF data model [*RDFPrimer*]. We also assume some familiarity with the HTTP protocol [*RFC2616*]. Wikipedia's article [*WP-HTTP*] serves as a good primer.

Table of Contents

1. Introduction

The Semantic Web is envisioned as a decentralised world-wide information space for sharing machine-readable data with a minimum of integration costs. Its two core challenges are the distributed modelling of the world with a shared data model, and the infrastructure where data and schemas can be published, found and used. Users benefit from getting information *"raw and now"* [Give] and in portable data formats [DP]. Providers often publish data embedded in a fixed user interface, in HTML. A basic question is thus how to publish information about resources in a way that allows interested users and software applications to find and interpret them.

On the Semantic Web, all information has to be expressed as *statements* about *resources*, like *the members of the company Example.com are Alice and Bob* or *Bob's telephone number is "+1 555 262"* or *this Web page was created by Alice*. Resources are identified by *Uniform Resource Identifiers* (URIs) [*RFC3986*]. This modelling approach is at the heart of *Resource Description Framework* (RDF) [*RDFPrimer*]. A nice introduction is given in the N3 primer [N3Primer].

Using RDF, the statements can be published on the Web site of the company. Others can read the data and publish their own information, linking to existing resources. This forms a distributed model of the world. It allows the user to pick any application to view and work with the same data, for example to see Alice's published address in your address book.

At the same time, Web documents have always been addressed with URIs (in common parlance often referred as Uniform Resource Locators, URLs). This is useful because it means we can easily make RDF statements about Web pages, but also dangerous because we can easily mix up Web pages and the things, or resources, described on the page.

So the question is, what URIs should we use in RDF? As an example, to identify the frontpage of the Web site of Example Inc., we may use `http://www.example.com/`. But what URI identifies the company as an organisation, not a Web site? Do we have to serve any content—HTML pages, RDF files—at those URIs? In this document we will answer these questions according to relevant specifications. We explain how to use URIs for things that are not Web pages, such as people, products, places, ideas and concepts such as ontology classes. We give detailed examples as to how the Semantic Web can (and should) be realised as a part of the Web.

2. URIs for Web Documents

Let us begin with an example. Assume that Example Inc., a fictional company producing "**Ex**treme Guitar **Ampl**ifi**er**s", has a Web site at `http://www.example.com/`. Part of the site is a white-pages service listing the names and contact details of the employees. Alice and Bob both work at Example Inc. The structure of the Web site might thus be:

`http://www.example.com/`
: the homepage of Example Inc.

`http://www.example.com/people/alice`
: the homepage of Alice

`http://www.example.com/people/bob`
: the homepage of Bob

Like everything on the traditional Web, each of the pages mentioned above are *Web documents*. Every Web document has its own URI. Note that a Web document is not the same as a file: a single Web document can be available in many different formats and languages, and a single file, for example a PHP script, may be responsible for generating a large number of Web documents with different URIs. A Web document is defined as something that has a URI and can return *representations* (responses in a format such as HTML or JPEG or RDF) of the identified resource in response to HTTP requests. In technical literature, such as *Architecture of the World Wide Web, Volume One* [*AWWW*], the term *Information Resource* is used instead of *Web document*.

On the traditional Web, URIs were used *primarily* for Web documents—to link to them, and to access them in a browser. The notion of resource *identity* was not so important on the traditional Web, a URL simply identified whatever we see when we type it into a browser.

2.1. HTTP and Content Negotiation

Web clients and servers use the HTTP protocol [*RFC2616*] to request representations of Web documents and send back the responses. HTTP has a powerful mechanism for offering different formats and language versions of the same Web document known as *content negotiation*.

When a user agent (such as a browser) makes an HTTP request, it sends along some HTTP headers to indicate what data formats and language it prefers. The server then selects the best match from its file system or generates the desired content on demand, and sends it back to the client. For example, a browser could send this HTTP request to indicate that it wants an HTML or XHTML representation of `http://www.example.com/people/alice` in English or German:

```
GET /people/alice HTTP/1.1
Host: www.example.com
Accept: text/html, application/xhtml+xml
Accept-Language: en, de
```

The server could answer:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Language: en
Content-Location: http://www.example.com/people.en.html
```

followed by the content of the HTML document in English.

Here we see Content negotiation [*TAG-Alt*] in action. The server interprets the `Accept-Language` headers in the request and decides to return the English representation of the resource in question. Note that the URI of this representation is passed back in the `Content-Location` header, this is not required but a recommended good practice (see [CHIPS], 7.2). Clients see that this URI is connected to the specific representation (in this case English) and search engines can refer to the different representations by using the different URIs. This implies that it is possible to have multiple representations of the same resource.

Content negotation is often implemented with a twist: Instead of a direct answer, the server *redirects* to another URL where the appropriate representation is found:

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

The redirect is indicated by a special *Status Code*, here `302 Found`. The client would now send another HTTP request to the new URL. By having separate URLs for different representations, this approach allows Web authors to link directly to a specific representation.

RDF/XML, the standard serialisation format of RDF, has its own content type, `application/rdf+xml`. Content negotiation thus allows publishers to serve HTML representations of a Web document to traditional Web browsers and RDF representations to Semantic Web-enabled user agents. This also allows servers to provide alternative RDF serialisation formats like Notation3 [*N3*] or TriX [*TriX*].

3. URIs for Real-World Objects

On the Semantic Web, URIs identify not just Web documents, but also real-world objects like people and cars, and even abstract ideas and non-existing things like a mythical unicorn. We call these *real-world objects* or *things*.

Given such a URI, how can we find out what it identifies? We need some way to answer this question, because otherwise it will be hard to achieve interoperability between independent information systems. We could imagine a service where we can look up a description of the identified resource, similar to today's search engines. But such a single point of failure is against the Web's decentralised nature.

Instead, we should use the Web itself—an extremely robust and scalable information publishing system—as a lookup service for resource descriptions. Whenever a URI is mentioned, we can look it up to retrieve a description containing relevant information and links to related data. This is so important that we make it our number one requirement for *cool* URIs:

> **1. Be on the Web.**
>
> Given only a URI, machines and people should be able to retrieve a description about the resource identified by the URI from the Web. Such a look-up mechanism is important to establish shared understanding of what a URI identifies. Machines should get RDF data and humans should get a readable representation, such as HTML. The standard Web transfer protocol, HTTP, should be used.

Let's assume Example Inc. wants to publish contact data of their employees on the Semantic Web so their business partners can import it into their address books. For example, the published data would contain these statements about Alice, written here in N3 syntax [*N3*]:

```
<URI-of-alice> a foaf:Person;
    foaf:name "Alice";
    foaf:mbox <mailto:alice@example.com>;
    foaf:homepage <http://www.example.com/people/alice> .
```

What URI should we use instead of the placeholder `<URI-of-alice>`? Certainly not `http://www.example.com/people/alice`, because that would confuse a person with a Web document, leading to misunderstandings: Is the homepage of Alice also named “Alice”? Can a homepage itself have

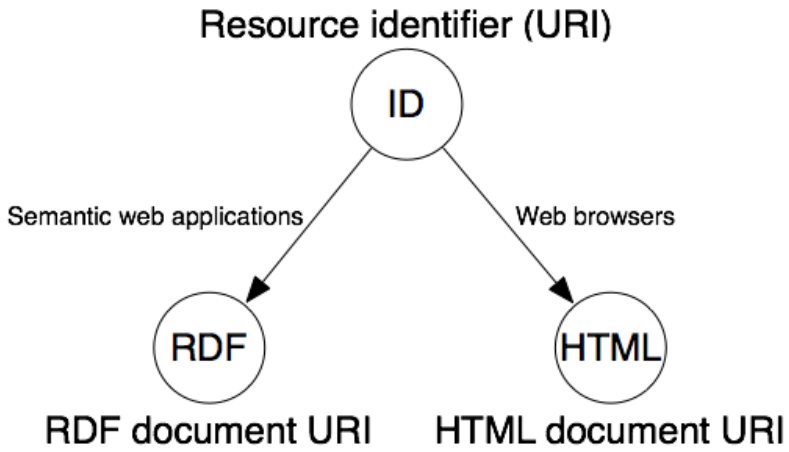
an e-mail address? And does it make sense for a home-page to have itself as its home-page? So we need another URI. (For in-depth treatments of this issue, see What HTTP URIs Identify? [*HTTP-URI2*] and Four Uses of a URL: Name, Concept, Web Location and Document Instance [*Booth*]).

Therefore our second requirement:

> **2. Be unambiguous.**
>
> There should be no confusion between identifiers for Web documents and identifiers for other resources. URIs are meant to identify only one of them, so one URI can't stand for both a Web document and a real-world object.

We note that our requirements seem to conflict with each other. If we can't use URIs of documents to identify real-world object, then how can we retrieve a representation about real-world objects based on their URI? The challenge is to find a solution that allows us to find the describing documents if we have just the resource's URI, using standard Web technologies.

The following picture shows the desired relationships between a resource and its representing documents:



3.1 Distinguishing between Representations and Descriptions

It is important to understand that using URIs, it is possible to identify both a thing (which may exist outside of the Web) and a Web document *describing* the thing. For example the person Alice is described on her homepage. Bob may not like the look of the *homepage*, but fancy the person Alice. So two URIs are needed, one for Alice, one for the homepage or a RDF document describing Alice. The question is where to draw the line between the case where either is possible and the case where *only* descriptions are available.

According to W3C guidelines ([AWWW], section 2.2.), we have a Web document (there called *information resource*) if *all its essential characteristics can be conveyed in a message*. Examples are a Web page, an image or a product catalog.

In HTTP, because a `200` response code should be sent when a Web document has been accessed, but a different setup is needed when publishing URIs that are meant to identify entities which are *not* Web documents.

In the next section, solutions are described that allow you to mint URIs for things and also allow clients to get a description of the thing using standard Web technologies.

4. Two Solutions

There are two solutions that meet our requirements for identifying real-world objects: *303 URIs* and *hash URIs*. Which one to use depends on the situation, both have advantages and disadvantages.

The solutions described in the following apply to deployment scenarios in which the RDF data and the HTML data is served separately, such as a standalone RDF/XML document along with an HTML document. The metadata can also be embedded in HTML, using technologies such as RDFa [RDFa Primer], microformats and other documents to which the GRDDL [GRDDL] mechanisms can be applied. In those cases the RDF data is extracted from the returned HTML document.

4.1. Hash URIs

The first solution is to use “hash URIs” for non-document resources. URIs can contain a *fragment*, a special part that is separated from the rest of the URI by a hash symbol (“#”).

When a client wants to retrieve a hash URI, then the HTTP protocol requires the fragment part to be stripped off before requesting the URI from the server. This means a URI that includes a hash cannot be retrieved directly, and therefore does not necessarily identify a Web document. But we can use them to identify other, non-document resources, without creating ambiguity.

If Example Inc. adopts this solution, then they could use these URIs to represent the company, Alice, and Bob:

`http://www.example.com/about#exampleinc`
: Example Inc., the company

`http://www.example.com/about#bob`
: Bob, the person

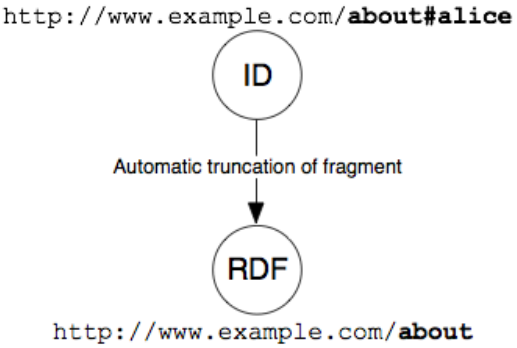
`http://www.example.com/about#alice`
: Alice, the person

Clients will always strip off the fragment part before requesting any of these URIs, resulting in a request to this URI:

`http://www.example.com/about`
: RDF document describing Example Inc., Bob, and Alice

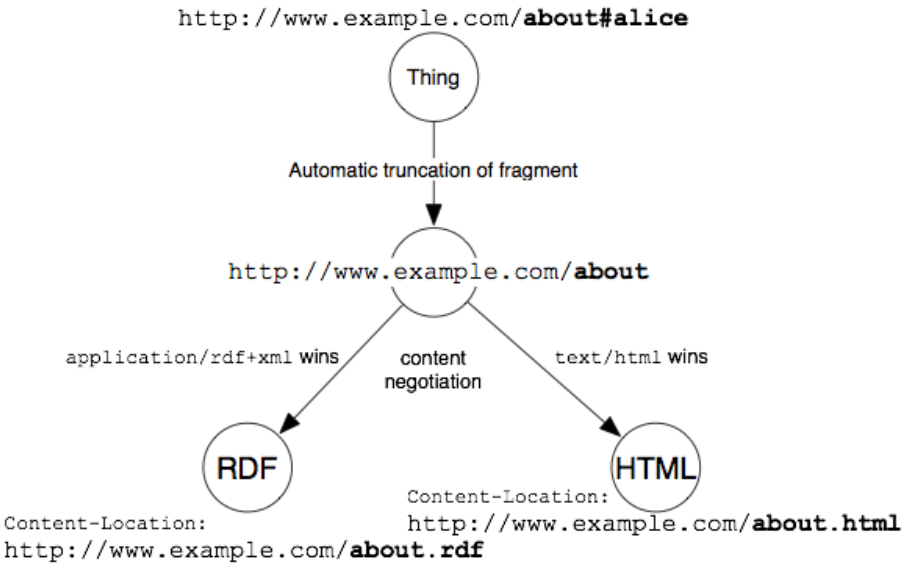
At this URI, Example Inc. could serve an RDF document that contains descriptions of all three resources, using the original hash URIs to identify the resources.

The following picture shows the hash URI approach without content negotiation:



Alternatively, content negotiation (see Section 2.1.) could be employed to redirect from the `about` URI to either a HTML or an RDF representation. The decision which to return is based on client preferences and server configuration, as explained below in Section 4.7. The `Content-Location` header should be set to indicate if the hash URI refers to a part of the HTML document or RDF document.

The following picture shows the hash URI approach with content negotiation:



4.2. 303 URIs forwarding to One Generic Document

The second solution is to use a special HTTP status code, `303 See Other`, to give an indication that the requested resource is not a regular Web document. Web architecture tells you that for a thing resource (URI) it is inappropriate to return a 200 because there is, in fact, no suitable representation for those resources. However, it is useful to provide information about those resources. The W3C's Technical Architecture Group proposes in its httpRange-14 resolution [*httpRange*] document a solution that is to direct you to a document which has information *about* the thing you asked about. By doing this we avoid ambiguity between the original, real-world object and the resource that represents it.

Since 303 is a redirect status code, the server can give the location of a document that represents the resource. If, on the other hand, a request is answered with one of the usual status codes in the 2XX range, like `200 OK`, then the client knows that the URI identifies a Web document.

If Example Inc. adopts this solution, they could use these URIs to represent the company, Alice and Bob:

`http://www.example.com/id/exampleinc`
: Example Inc., the company

`http://www.example.com/id/bob`
: Bob, the person

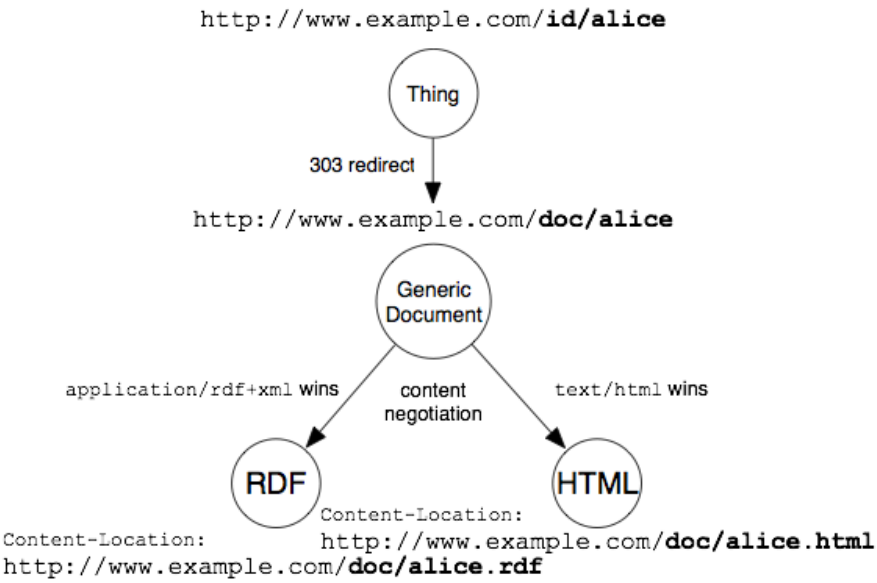
`http://www.example.com/id/alice`
: Alice, the person

The Web server would be configured to answer requests to all these URIs with a 303 status code and a `Location` HTTP header that provides the URL of a document that represents the resource. For example, to redirect from `http://www.example.com/id/alice` to `http://www.example.com/doc/alice`.

Content-negotiation is then used when retrieving a representation from the document URI using a HTTP request. The server decides (see Section 4.7) to return either HTML or RDF (or more alternative forms) and sets the `Content-Location` header to the URI where the specific representation can be retrieved.

This setup should be used when the RDF and HTML (and possibly more alternative representations) convey the *same information in different forms*. When the information in the variations differs considerably, the 303 approach as described below should be used.

See the following illustration for the solution providing the generic document URI.

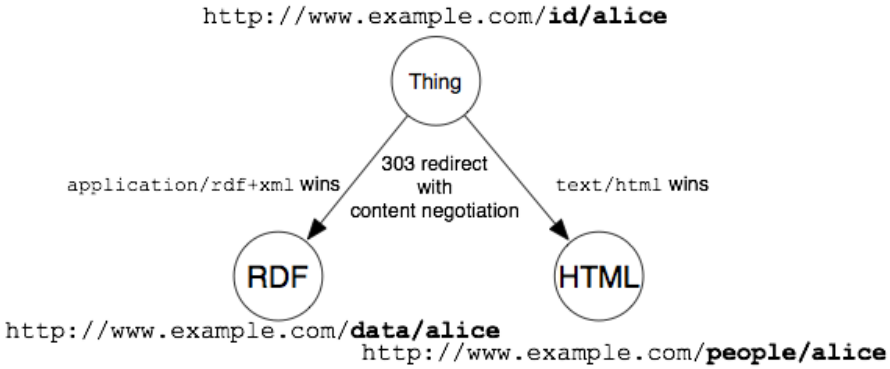


In this setup, the server forwards from the identification URI to the generic document URI. This has the advantage that clients can bookmark and further work with the generic document. A user having a RDF-capable client could bookmark the document, and mail it to another user (or device) which then dereferences it and gets the HTML *or* the RDF view. Also, the server can add representations in new languages in the future. Just because the client started with the URI of a thing, it doesn't mean that the document involved is not a first class document on the WWW. The background of generic document resources is described in [GenRes].

4.3. 303 URIs forwarding to Different Documents

When the RDF and HTML representations of the resource differ substantially, the previous setup should not be used. They are not two versions of the same document, but different documents altogether. Again, the Web server would be configured to answer requests with a 303 status code and a `Location` HTTP header that provides the URL of a document that represents the resource.

The following picture shows the redirects for the 303 URI solution without the generic document URI:



The server could employ content negotiation (see Section 2.1.) to send either the URL of an HTML description or RDF. HTTP requests for HTML content would be redirected to the HTML URLs we gave in Section 2. Requests for RDF data would be redirected to RDF documents, such as:

`http://www.example.com/data/exampleinc`
: RDF document describing Example Inc., the company

`http://www.example.com/data/bob`
: RDF document describing Bob, the person

`http://www.example.com/data/alice`
: RDF document describing Alice, the person

Each of the RDF documents would contain statements about the appropriate resource, using the original

URI, e.g. `http://www.example.com/id/alice`, to identify the described resource.

4.4. Choosing between 303 and Hash

Which approach is better? It depends. The hash URIs have the advantage of reducing the number of necessary HTTP round-trips, which in turn reduces access latency. A family of URIs can share the same non-hash part. The descriptions of `http://www.example.com/about#exampleinc`, `http://www.example.com/about#alice`, and `http://www.example.com/about#bob` are retrieved with a single request to `http://www.example.com/about`. However this approach has a downside. A client interested only in `#product123` will inadvertently load the data for all other resources as well, because they are in the same file. 303 URIs, on the other hand, are very flexible because the redirection target can be configured separately for each resource. There could be one describing document for each resource, or one large document for all of them, or any combination in between. It is also possible to change the policy later on.

When using 303 URIs for an ontology, like FOAF, network delay can reduce a client's performance considerable. The large number of redirects may cause higher latency. A client looking up a set of terms through 303 may use many requests, even though the first request has already loaded everything there is to know.

When hosting large-scale datasets with the 303 solution, clients may be tempted to download all data using many requests. We advise to additionally provide SPARQL endpoints or comparable services to answer complex queries on the server directly, rather than to let the client download a large set of data via HTTP.

Note also, that both *303 and Hash can be combined*, allowing a large dataset to be separated into multiple parts and have an identifier for a non-document resource. An example for a combination of 303 and Hash is:

`http://www.example.com/bob#this`
: Bob, the person with a combined URI.

Any fragment identifier is valid, `this` in the above URI is a suggestion you may want to copy for your implementations.

> **Conclusion.**
>
> Hash URIs should be preferred for rather small and stable sets of resources that evolve together. The ideal case are RDF Schema vocabularies and OWL ontologies, where the terms are often used together, and the number of terms is unlikely to grow out of control in the future.
>
> Hash URIs without content negotiation can be implemented by simply uploading static RDF files to a Web server, without any special server configuration. This makes them popular for quick-and-dirty RDF publication.
>
> URIs of the `bob#this` form can be used for large sets of data that are, or may grow, beyond the point where it is practical to serve all related resources in a single document. 303 URIs may also be used for such data sets, making neater-looking URIs, but with an impact on run-time performance and server load.
>
> If in doubt, follow your nose.

4.5. Cool URIs

The best resource identifiers don't just provide descriptions for people and machines, but are designed with simplicity, stability and manageability in mind, as explained by Tim Berners-Lee in *Cool URIs don't change* and by the W3C Team in *Common HTTP Implementation Problems* (sections 1 and 3):

Simplicity.
: Short, mnemonic URIs will not break as easily when sent in emails and are in general easier to remember, e.g. when debugging your Semantic Web server.

Stability.

Once you set up a URI to identify a certain resource, it should remain this way as long as possible. Think about the next ten years. Maybe twenty. Keep implementation-specific bits and pieces such as `.php` and `.asp` out of your URIs, you may want to change technologies later.

Manageability.
Issue your URIs in a way that you can manage. One good practice is to include the current year in the URI path, so that you can change the URI-schema each year without breaking older URIs. Keeping all 303 URIs on a dedicated subdomain, e.g. `http://id.example.com/alice`, eases later migration of the URI-handling subsystem.

4.6. Linking

All the URIs related to a single real-world object—resource identifier, RDF document URL, HTML document URL—should also be explicitly linked with each other to help information consumers understand their relation. For example, in the 303 URI solution for Example Inc., there are three URIs related to Alice:

`http://www.example.com/id/alice`
Identifier for Alice, the person
`http://www.example.com/people/alice`
Alice's homepage
`http://www.example.com/data/alice`
RDF document with description of Alice

Two of them are Web document URLs. The RDF document located at `http://www.example.com/data/alice` might contain these statements (expressed in N3):

```
<http://www.example.com/id/alice>
    foaf:page <http://www.example.com/people/alice>;
    rdfs:isDefinedBy <http://www.example.com/data/alice>;

    a foaf:Person;
    foaf:name "Alice";
    foaf:mbox <mailto:alice@example.com>;
    ...
```

The document makes statements about Alice, the person, using the resource identifier. The first two properties relate the resource identifier to the two document URIs. The `foaf:page` statement links it to the HTML document. This allows RDF-aware clients to find a human-readable resource, and at the same time, by linking the page to its topic, defines useful metadata about that HTML document. The `rdfs:isDefinedBy` statement links the person to the document containing its RDF description and allows RDF browsers to distinguish this main resource from other auxiliary resources that just happen to be mentioned in the document. We use `rdfs:isDefinedBy` instead of its weaker superproperty `rdfs:seeAlso` because the content at `/data/alice` is authoritative. The remaining statements are the actual white pages data.

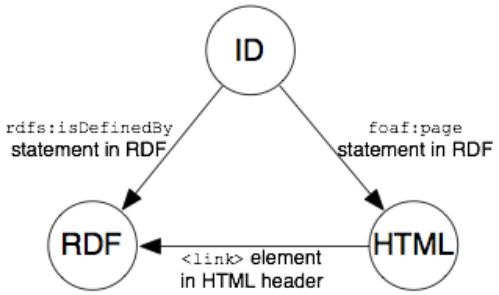
The HTML document at `http://www.example.com/people/alice` should contain in its header a `<link>` element that points to the corresponding RDF document:

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
  <head>
    <title>Alice's Homepage</title>
    <link rel="alternate" type="application/rdf+xml"
          title="RDF Representation"
          href="http://www.example.com/data/alice" />
  </head> ...
```

This allows RDF-aware Web clients to discover the RDF information. The approach is recommended in the RDF/XML specification ([*RDFXML*], section 9). If the RDF data is *about* the Web page, rather than an expression of the information in it, then we recommend using `rel="meta"` instead of `rel="alternate"`.

The client also can deduce similar link information directly from the HTTP headers: that a thing is described by a Web document which can be found at the end of a 303 redirect; that the `Content-Location` resource is a content-specific version of the generic document, and more. Ontologies for these relations are not discussed here.

The following illustration shows how the RDF and HTML documents should relate the three URIs to each other:



4.7. Implementing Content Negotiation

The W3C's Semantic Web Best Practices and Deployment Working Group has published a document that describes how to implement the solutions presented here on the Apache Web server. The *Best Practice Recipes for Publishing RDF Vocabularies* [*Recipes*] mostly discuss the publication of *RDF vocabularies*, but the ideas can also be applied to other kinds of small RDF datasets that are published from static files.

However, especially when it comes to content negotiation, the Recipes document doesn't cover some important details. Content negotiation is a bit more difficult in practice because of mixed-mode clients that can deal with both HTML and RDF, such as Firefox with the Tabulator extension.

These browsers announce their ability to consume both RDF and HTML through `Accept` headers that use `q` (quality) values:

```
Accept: application/rdf+xml;q=0.7, text/html
```

This browser accepts RDF with a `q` value of 0.7 and HTML with a `q` value of 1.0 (the default). This means the browser has a slight preference for HTML over RDF.

Now, a client preference for HTML doesn't necessarily mean that every server should send HTML. The server has to look at the client's preferences, and then it must make a decision based on the quality of the different variants it could offer. For example:

- If the HTML variant is a simple low-quality rendering of the RDF, like a property-value table or a list of triples, then the server should send the RDF, unless the client has a very strong preference for HTML.
- If HTML and RDF variant contain the same information, and both are of high quality, then the server should treat both variants with equal preference, and leave the choice to the client's preferences.
- If the RDF variant is only a part of the information offered in the HTML, or is scraped from the HTML, then the server should probably send the HTML, unless the client has a strong preference for RDF.

There are algorithms for choosing the best match by comparing client preferences with the quality of the server's available variants. For example, the Apache server can be configured with server-side `qs` values that specify their relative quality.

A `qs` value of 1.0 for `application/rdf+xml` and 0.5 for `text/html`, would mean that the HTML variant has only approximately half the quality of the RDF and might be appropriate in the first case from the list above. If the HTML is a news article and the RDF contains just minimal information such as title, date and author, then 1.0 for the HTML and 0.1 for the RDF would be appropriate.

To determine the best variant for a particular client, Apache multiplies the client's `q` value for HTML with the configured `qs` value for HTML; and the same for RDF. The variant with the higher number wins. Apache's documentation has a section with a detailed description of its content negotiation algorithm [ApCN]. HTTP's `Accept` header is described in detail in section 14.1 of the HTTP specification [HTTP-SPEC].

Content negotiation, with all its details, is fairly complex, but it is a powerful way of choosing the best

variant for mixed-mode clients that can deal with HTML and RDF.

5. Examples from the Web

Not all projects that work with Semantic Web technologies make their data available on the Web. But a growing number of projects follow the practices described here. This section gives a few examples.

ECS Southampton. The School of Electronics and Computer Science at University of Southampton has a Semantic Web site that employs the 303 solution and is a great example of Semantic Web engineering. It is documented in the *ECS URI System Specification* [*ECS*]. Separate subdomains are used for HTML documents, RDF documents, and resource identifiers. Take these examples:

`http://id.ecs.soton.ac.uk/person/1650`
: URI for Wendy Hall, the person

`http://www.ecs.soton.ac.uk/people/wh`
: HTML page about Wendy Hall

`http://rdf.ecs.soton.ac.uk/person/1650`
: RDF about Wendy Hall

Entering the first URI into a normal Web browser redirects to an HTML page about Wendy Hall. It presents a Web view of all available data on her. The page also links to her URI and to her RDF document.

D2R Server is an open-source application that can be used to publish data from relational databases on the Semantic Web in accordance with these guidelines. It employs the 303 solution and content negotiation. For example, the *D2R Server publishing the DBLP Bibliography Database* publishes several thousand bibliographical records and information about their authors. Example URIs, again connected via 303 redirects:

`http://www4.wiwiss.fu-berlin.de/dblp/resource/person/315759`
: URI for Chris Bizer, the person

`http://www4.wiwiss.fu-berlin.de/dblp/page/person/315759`
: HTML page about Chris Bizer

The RDF document for Chris Bizer is a SPARQL query result from the server's SPARQL endpoint:

```
http://www4.wiwiss.fu-berlin.de/dblp/sparql?query=
DESCRIBE+\%3Chttp\%3A\%2F\%2Fwww4.wiwiss.fu-berlin.de
\%2Fdblp\%2Fresource\%2Fperson\%2F315759\%3E
```

The SPARQL query encoded in this URI is:

```
DESCRIBE <http://www4.wiwiss.fu-berlin.de/dblp/resource/person/315759>
```

This shows how a SPARQL endpoint can be used as a convenient method of serving resource descriptions.

Semantic MediaWiki is an open-source Semantic wiki engine. Authors can use special wiki syntax to put semantic attributes and relationships into wiki articles. For each article, the software generates a 303 URI that identifies the article's topic, and serves RDF descriptions generated from the attributes and relationships. Semantic MediaWiki drives the OntoWorld wiki. It has an article about the city of Karlsruhe:

`http://ontoworld.org/wiki/Karlsruhe`
: the article, an HTML document

`http://ontoworld.org/wiki/_Karlsruhe`
: the city of Karlsruhe

`http://ontoworld.org/wiki/Special:ExportRDF/Karlsruhe`
: RDF description of Karlsruhe

The URI of the RDF description is less than ideal, because it exposes the implementation (php) and refers redundantly to RDF in the path and in the query. A much cooler URI would be for example `http://ontoworld.org/data/Karlsruhe`, as it allows content negotiation to be used to serve the data in

RDF, RIF (Rule Interchange Format), or whatever else we think of next.

6. Other Resource Naming Proposals

Many other approaches have been suggested over the years. While most of them are appropriate in special circumstances, we feel that they do not fit the criteria from Section 3, which are to *be on the Web* and *don't be ambiguous*. Therefore they are not adequate as general solutions for building a standards-based, non-fragmented, decentralized Semantic Web. We will discuss two of these approaches in some detail.

6.1. New URI Schemes

HTTP URIs already identify Web resources and Web documents, not other kinds of resources. Shouldn't we create a new URI scheme to identify other resources? Then we could easily distinguish them from Web documents just by looking at the first characters of the URI. For example, the *info* scheme can be used to identify books based on a LCCN number: `info:lccn/2002022641`.

Here are examples of such new URI schemes. A longer list is provided by Thompson and Orchard in *URNs, Namespaces and Registries* [*TAG-URNs*].

- **Magnet** is an open URI scheme enabling seamless integration between Web sites and locally-running utilities, such as file-management tools. It is based on hash-values, a URI looks like this: `magnet:?xt=urn:sha1:YNCKHTQCWBTRNJIV4WNAE52SJUQCZO5C`.
- The **`info:` URI scheme** is proposed to identify information assets that have identifiers in existing public namespaces. Examples are URIs for LCCN numbers (`info:lccn/2002022641`) and the Dewey decimal system (`info:ddc/22/eng//004.678`).
- The idea of **Tag URIs** is to generate collision-free URIs by using a domain name and the date when the URI was allocated. Even if the domain changes ownership at a later date, the URI remains unambiguous. Example: `tag:hawke.org,2001-06-05:Taiko`.
- **XRI** defines a scheme and resolution protocol for abstract identifiers. The idea is to use URIs that contain wildcards, to adapt to changes of organizations, servers, etc.
  Examples are `@Jones.and.Company/(+phone.number)` or `xri://northgate.library.example.com/(urn:isbn:0-395-36341-1)`.

To be truly useful, a new scheme must be accompanied by a protocol defining how to access more information about the identified resource. For example, the `ftp://` URI scheme identifies resources (files on an FTP server), and also comes with a protocol for accessing them (the FTP protocol).

Some of the new URI schemes provide no such protocol at all. Others provide a Web Service that allows retrieval of descriptions using the HTTP protocol. The identifier is passed to the service, which looks up the information in a central database or in a federated way. The problem here is that a failure in this service renders the system unusable.

Another drawback can be a dependence on a standardization body. To register new parts in the `info:` space, a standardization body has to be contacted. This, or paying a license fee before creating a new URI, slows down adoption. In such cases a standardization body is desirable to ensure that all URIs are unique (e.g. with ISBNs). But this can be achieved using HTTP URIs inside an HTTP namespace owned and managed by the standardization organization.

Independent of standardization body and retrievability, pending patents and legal issues can influence the adoption of a new URI scheme. When using patented technology, implementers should verify that a Royalty-Free license is available.

The problems with new URI schemes are discussed at length in *URNs, Namespaces and Registries*.

6.2. Reference by Description

"Reference by Description" radically solves the URI problem by doing away with URIs altogether: Instead of *naming* resources with a URI, *anonymous nodes* are used, and are *described* with information that allows us to find the right one. A person, for example, could be described by name, date of birth, and social security number. These pieces of information should be sufficient to uniquely identify a person.

A popular practice is the use of a person's email address as a uniquely identifying piece of information. The `foaf:mbox` property is used in Friend of a Friend (*FOAF*) profiles for this purpose. In OWL, this kind

of property is known as an *Inverse Functional Property* (IFP). When an agent encounters two resources with the same email address, it can infer that both refer to the same person and can treat them as one.

But how to *be on the Web* with this approach? How to enable agents to download more data about resources we mention? There is a best practice to achieve this goal: Provide not only the IFP of the resource (e.g. the person's email address), but also an `rdfs:seeAlso` property that points to a Web address of an RDF document with further information about it. We see that HTTP URIs are still used to identify the location where more information can be downloaded.

Furthermore, we now need several pieces of information to refer to a resource, the IFP value and the RDF document location. The simple act of linking by using a URI has become a process involving several moving parts, and this increases the risk of broken links and makes implementation more cumbersome.

Regarding FOAF's practice of avoiding URIs for people, we agree with Tim Berners-Lee's advice: “Go ahead and give yourself a URI. You deserve it!”

7. Conclusion

Resource names on the Semantic Web should fulfill two requirements: First, a description of the identified resource should be retrievable with standard Web technologies. Second, a naming scheme should not confuse things and the documents representing them.

We have described two approaches that fulfill these requirements, both based on the HTTP URI scheme and protocol. One is to use the 303 HTTP status code to redirect from the resource identifier to the describing document. One is to use “hash URIs” to identify resources, exploiting the fact that hash URIs are retrieved by dropping the part after the hash and retrieving the other part.

The requirement to distinguish between resources and their descriptions increases the need for coordination between multiple URIs. Some useful techniques are: embedding links to RDF data in HTML documents, using RDF statements to describe the relationship between the URIs, and using content negotiation to redirect to an appropriate description of a resource.

8. Acknowledgements

Many thanks to Tim Berners-Lee who invested much time and helped us understanding the TAG solution by answering chat requests and contributing many emails with clarifications and detailled reviews of this document. Special thanks go to Stuart Williams, Norman Walsh and all the other members from TAG, who reviewed this document and provided essential feedback in June 2007 and September 2007 about many formulations that were (accidentially) contrary to the TAG's view. Also special thanks to the Semantic Web Deployment Group's members Michael Hausenblas, Vit Novacek, and Ed Summers' reviews and their review summary sent in October 2007. We wish to thank everyone else who has reviewed drafts of this document, especially Chris Bizer, Gunnar AAstrand Grimnes, Harry Halpin, Xiaoshu Wang, Henry S. Thompson, Jonathan Rees, and Christoph Päper. Susie Stephens reviewed the document, managed SWEO, and helped us to stay on track. Ivan Herman did much to verify that the W3C requirements are met and submitted the note.

This work was supported by the German Federal Ministry of Education, Science, Research and Technology (BMBF), (Grants 01 IW C01, Project EPOS: Evolving Personal to Organizational Memories; and 01 AK 702B, Project InterVal: Internet and Value Chains) and by the European Union IST fund (Grant FP6-027705, Project Nepomuk).

9. References

[AWWW]
Architecture of the World Wide Web, Volume One, Ian Jacobs, Norman Walsh, Editors. World Wide Web Consortium, 15 December 2004. This edition is http://www.w3.org/TR/2004/REC-webarch-20041215/. The latest edition is available at http://www.w3.org/TR/webarch/.

[ApCN]
Apache HTTP Server Version 2.0 Documentation, Chapter Content Negotiation. This document is available at http://httpd.apache.org/docs/2.0/content-negotiation.html.

[Booth]
Four Uses of a URL: Name, Concept, Web Location and Document Instance, David Booth. 28 January 2003. This document is available at http://www.w3.org/2002/11/dbooth-names/dbooth-names_clean.htm.

[CHIPS]
Common HTTP Implementation Problems, Olivier Théreaux, Editor. World Wide Web Consortium, 28 January 2003. This edition is http://www.w3.org/TR/2003/NOTE-chips-20030128/. The latest edition is available at http://www.w3.org/TR/chips/.

[Cool]
Cool URIs don't change, Tim Berners-Lee, 1998. This document is available at http://www.w3.org/Provider/Style/URI.

[DP]
The DataPortability Project. http://dataportability.org/

[ECS]
ECS URI System Specification, Colin Williams, Nick Gibbins. ECS Southampton, 2006. This document is available at http://id.ecs.soton.ac.uk/docs/.

[FOAF]
FOAF Vocabulary Specification 0.9, Dan Brickley, Libby Miller. 24 May 2007. This edition is http://xmlns.com/foaf/spec/20070524.html. The latest edition is available at http://xmlns.com/foaf/spec/.

[Give]
Give Us the Data Raw, and Give it to Us Now. Rufus Pollock. 7th November 2007.

[GenRes]
Generic Resources, Tim Berners-Lee. This document is available at http://www.w3.org/DesignIssues/Generic.html.

[GRDDL]
Gleaning Resource Descriptions from Dialects of Languages (GRDDL), Dan Connolly, Editor, W3C Recommendation 11 September 2007. This edition is http://www.w3.org/TR/2007/REC-grddl-20070911/. The latest edition is available at http://www.w3.org/TR/grddl/.

[HTTP-URI2]
What HTTP URIs Identify, Tim Berners-Lee. 9 June 2005. This document is available at http://www.w3.org/DesignIssues/HTTP-URI2.html.

[httpRange]
[httpRange-14] Resolved, Roy Fielding. 18 June 2005. This archived www-tag email message is available at http://lists.w3.org/Archives/Public/www-tag/2005Jun/0039.html.

[HTTP-SPEC]
RFC2616, Hypertext Transfer Protocol -- HTTP/1.1, http://www.rfc.net/rfc2616.html#s14.1

[N3]
Notation 3, Tim Berners-Lee, Dan Connolly, 2008. This document is available at http://www.w3.org/TeamSubmission/n3/.

[N3Primer]
Primer: Getting into RDF & Semantic Web using N3. Tim Berners-Lee, 2005. http://www.w3.org/2000/10/swap/Primer

[RDFa Primer]
RDFa Primer 1.0 - Embedding Structured Data in Web Pages (see http://www.w3.org/2006/07/SWD/RDFa/primer/.)

[RDFPrimer]
RDF Primer, Frank Manola, Eric Miller, Editors. World Wide Web Consortium, 10 February 2004. This edition is http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. The latest edition is available at http://www.w3.org/TR/rdf-primer/.

[RDFXML]
RDF/XML Syntax Specification (Revised), Dave Beckett, Editor. World Wide Web Consortium, 10 February 2004. This edition is http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. The latest edition is available at http://www.w3.org/TR/rdf-syntax-grammar/.

[Recipes]
Best Practice Recipes for Publishing RDF Vocabularies, Alistair Miles, Thomas Baker, Ralph Swick, Editors. World Wide Web Consortium, 23 January 2008. This edition is http://www.w3.org/TR/2008/WD-swbp-vocab-pub-20080123/. It is a work in progress. The latest edition is available at http://www.w3.org/TR/swbp-vocab-pub/.

[RFC2616]
RFC 2616: Hypertext Transfer Protocol - HTTP/1.1, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. IETF, 1999. This document is available at http://www.ietf.org/rfc/rfc2616.txt.

[RFC3986]
RFC 3986: Uniform Resource Identifier (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter. IETF, 2005. This document is available at http://www.ietf.org/rfc/rfc3986.txt.

[SMW]
Semantic Wikipedia, Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, Rudi Studer. University of Karlsruhe, 2006. This document is available at http://www.aifb.uni-karlsruhe.de/WBS/hha/papers/SemanticWikipedia.pdf.

[TAG-Alt]

On Linking Alternative Representations To Enable Discovery And Publishing, T.V. Raman. World Wide Web Consortium, 1 November 2006. This edition is http://www.w3.org/2001/tag/doc/alternatives-discovery-20061101.html. The latest edition is available at http://www.w3.org/2001/tag/doc/alternatives-discovery.html.

[TAG-URNs]
URNs, Namespaces and Registries, Henry S. Thompson, David Orchard. World Wide Web Consortium, 17 August 2006. This edition is http://www.w3.org/2001/tag/doc/URNsAndRegistries-50-2006-08-17.html. It is a work in progress. The latest edition is available at http://www.w3.org/2001/tag/doc/URNsAndRegistries-50.html.

[TriX]
RDF Triples in XML, Jeremy J. Carroll, Patrick Stickler, 2004. This document is available at http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Stickler01/EML2004Stickler01.html.

[WP-HTTP]
Hypertext Transfer Protocol, Wikipedia contributors. Wikipedia, 8 October 2007. The latest version of this document is available at http://en.wikipedia.org/wiki/HTTP.

10. Change log

29 November 2006
1.0 Initial Version.

9 August 2007
1.1 Revised Version. Changes based on TAG review.

28 November 2007
Leo Sauermann included more feedback from reviews contributed by TAG, SWD, and Tim Berners-Lee.

8 December 2007
Danny Ayers did proofreading, minor grammar/idiomatic/editorial changes (I've tried not to make any changes that substantively modify the content, though some come close...). XHMTL validated with nxml-mode emacs

12 December 2007
Leo Sauermann included link to GRDDL as suggested by Danny Ayers, minor changes of todo notes. Document was remodelled to Working Draft status - all feedback by SWD, TAG, and Tim Berners Lee either has been addressed or is listed in this document as todos using @@-symbols and the css class "todo".

17 December 2007
Document published as Working Draft at http://www.w3.org/TR/2007/WD-cooluris-20071217/

23 Februar 2008
All feedback received on Working Draft.

20 March 2008
All feedback incorporated, issues are listed and addressed in this document.

21 March 2008
Document published as Last Call Working Draft at http://www.w3.org/TR/2008/WD-cooluris-20080321/

31 March 2008
Document published as Interest Group Note. Feedback to previous version and changes are listed here.

Thèse de Roy T. Fielding - Traduction du Chapitre 5 : REST

Statut du document traduit

Ceci est une traduction du chapitre 5 de la thèse de Roy T. Fielding.

Il ne s'agit pas de la version officielle mais d'une traduction destinée aux francophones afin de mieux comprendre l'original. Le document complet original,

«Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.»

est disponible à l'adresse http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

La traduction a été réalisée en respectant les droits de reproduction définis par auteur, consultables dans la FAQ du document d'origine.

Avertissement

Des erreurs ont pu survenir malgré le soin apporté à ce travail. Si vous décelez une erreur, n'hésitez pas à me contacter.

Notes sur la traduction

Certains concepts sont difficiles à exprimer en français, ou peuvent bénéficier d'une explication. Par endroits, les expressions originales en anglais viennent en renfort dans le texte sous cette forme :
ex. traduction *[ndt. translation]*

- Adresse : `<http://www.opikanoba.org/tr/fielding/rest/>`
- Traducteur : Frédéric Laurent (`fl at opikanoba.org`)
- Correcteur(s) :
  - Jean-Claude Simon (`jean-claude.simon at enesad.fr`)
  - Emmanuel Benoit (`emmanuel.benoit at opikanoba.org`)
- Date de traduction : 26 juillet 2006
- Dernière mise à jour : **05 août 2006**

Representational State Transfer (REST)

Ce chapitre présente le modèle d'architecture REST (Representational State Transfer) pour les systèmes d'hypermedia répartis. Il décrit les principes logiciels sur lesquels s'appuie REST et les contraintes, liées aux interactions, retenues pour maintenir ces principes. Par ailleurs, ces contraintes sont confrontées à celles d'autres modèles d'architecture. REST est un modèle hybride dérivé de plusieurs modèles basés sur les concepts réseau, décrits dans le chapitre 3, et combiné avec des contraintes supplémentaires qui définissent une interface uniforme de connecteur. Le framework d'architecture logicielle décrit au chapitre 1 est utilisé pour définir les éléments de REST et pour examiner des exemples de processus, de connecteurs et de vues de données des architectures prototypées.

5.1 Dériver REST

Le principe de conception de l'architecture du Web peut être décrit par un modèle comprenant l'ensemble des contraintes appliquées aux éléments de cette architecture. En examinant l'impact de chaque contrainte lors de son ajout au modèle, nous pouvons identifier les propriétés induites par les contraintes du Web. Des contraintes complémentaires peuvent alors être appliquées pour former un nouveau modèle d'architecture. Celui-ci reflétera mieux les propriétés que l'on souhaite qu'une architecture moderne du Web possède. Cette section donne une vue d'ensemble de REST en déroulant un processus qui va le dériver comme un modèle d'architecture. Les sections suivantes décriront plus en détail les contraintes spécifiques qui composent le modèle REST.

5.1.1 Commençons par le modèle vide *[ndt. null]*

Il existe deux façons classiques d'aborder un processus de conception d'une architecture, qu'il soit appliqué au bâtiment ou au logiciel. La première est qu'un concepteur parte de zéro -- une feuille blanche, un tableau blanc -- et construise petit à petit une architecture à base de composants connus jusqu'à ce que cette architecture satisfasse les besoins du système envisagé. La seconde approche consiste à commencer par les besoins du système dans son ensemble, sans contraintes. Puis, de façon incrémentale, les contraintes sont identifiées et appliquées aux éléments du système afin de différencier l'espace de conception et de permettre aux forces qui influencent son comportement de s'arranger naturellement, en harmonie avec le système. Le premier procédé met en avant la créativité et une imagination sans limite. Le second met l'accent sur les contraintes et la compréhension du contexte du système. REST a été développé en utilisant le second processus. Les schémas 5-1 à 5-8 l'illustrent graphiquement, en montrant comment les contraintes appliquées au fur et à mesure pourraient différencier la vue processus de l'architecture.

Le modèle vide *[ndt.null]* (schéma 5-1) représente simplement un ensemble vide de contraintes. D'un point de vue architectural, le modèle vide décrit un système dans lequel aucune frontière distincte n'existe entre les composants. C'est le point de départ pour notre description de REST.

Figure 5-1. Null Style

Schéma 5-1: Le modèle vide

5.1.2 Client-Serveur

Les premières contraintes que l'on ajoute à notre modèle hybride sont celles du modèle architectural client/serveur (schéma 5-2), décrit dans la section 3.4.1. La séparation des concepts est le principe sous-jacent du client/serveur. En séparant les problématiques d'interface utilisateur de celles du stockage de données, nous améliorons la portabilité de l'interface utilisateur à travers les diverses plates-formes. Nous améliorons par ailleurs la possibilité de montée en charge en simplifiant les composants du serveur. Mais l'aspect sans doute le plus important pour le Web est que cette séparation permet aux composants d'évoluer indépendamment. De ce fait, ce modèle s'adapte aux larges besoins (à l'échelle d'Internet) des multiples domaines organisationnels.

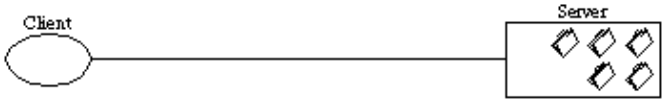


Figure 5-2. Client-Server

Schéma 5-2: Le modèle client/serveur

5.1.3 Sans état (Stateless)

Nous ajoutons ensuite une contrainte à l'interaction entre le client et le serveur : la communication doit être par nature sans état, comme dans le modèle «client/serveur sans état» *[ndt. CSS pour client-stateles-server]* de la section 3.4.3 (schéma 5-3). Ainsi chaque requête du client vers le serveur doit contenir toutes les informations nécessaires pour que cette demande soit comprise, et elle ne peut tirer profit d'aucun contexte stocké sur le serveur. L'état de la session est donc entièrement détenu par le client.

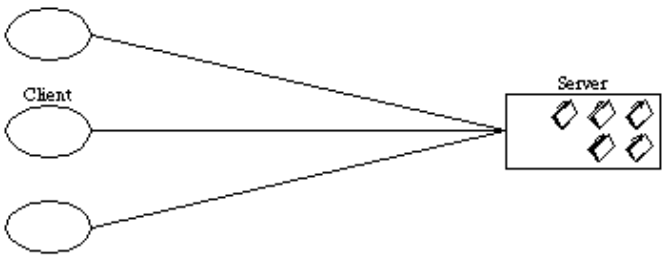


Figure 5-3. Client-Stateless-Server

Schéma 5-3: Le modèle client-serveur-sans état

Cette contrainte induit les propriétés suivantes : visibilité, fiabilité et faculté de montée en charge. La visibilité est améliorée car un système de supervision n'a pas besoin de regarder au delà des simples informations de la requête afin d'en déterminer sa nature complète. La fiabilité est améliorée car il est plus simple de faire face (recovery) à des échecs partiels [133]. Enfin, la possibilité de montée en

charge est meilleure, car le fait de ne pas stocker l'état entre les requêtes permet aux composants du serveur de libérer rapidement les ressources. De plus, la mise en œuvre est grandement simplifiée car le serveur n'a pas besoin de gérer l'utilisation des ressources à travers les différentes requêtes.

Comme la plupart des choix d'architecture, la contrainte sans état fait apparaître un compromis quant à la conception. Son inconvénient est qu'elle peut diminuer les performances réseau en répétant les données (surplus par interaction) envoyées dans une série de requêtes, puisqu'elles ne peuvent pas être conservées sur le serveur dans un contexte partagé. En outre, laisser l'état de l'application côté client réduit le contrôle qu'a le serveur sur le comportement cohérent de l'application, puisqu'elle devient dépendante de la mise en oeuvre correcte de la sémantique au travers des multiples versions de client.

5.1.4 Cache

Afin d'améliorer les performances réseau, nous ajoutons des contraintes de cache afin de former le modèle «client/serveur sans état avec cache» de la section 3.4.4 (schéma 5-4). Les contraintes de cache imposent que les données d'une réponse à une requête soient marquées, implicitement ou explicitement, comme pouvant être mises ou non en cache. Si une réponse peut être mise en cache *[ndt. «cacheable»]*, alors le cache du client obtient le droit de réutiliser ces données de réponse pour des demandes ultérieures équivalentes.

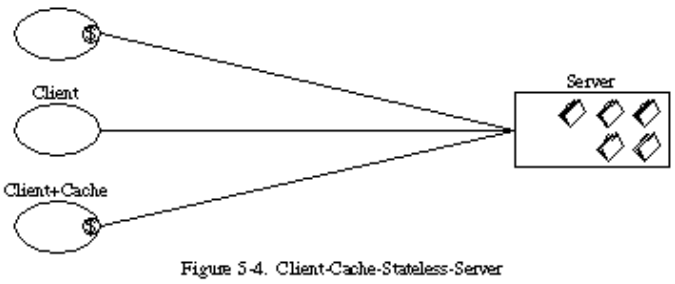


Figure 5-4. Client-Cache-Stateless-Server

Schéma 5-4: Le modèle client/serveur sans état avec cache

L'ajout de contraintes de cache a l'avantage d'offrir la possibilité d'éliminer tout ou partie de certaines interactions, améliorant ainsi l'efficacité, la montée en charge et la perception de performance qu'a l'utilisateur, en réduisant la latence moyenne dans une série d'interactions. Le compromis est qu'un cache peut diminuer la fiabilité si les données obsolètes qui y sont présentes diffèrent de manière significative des données qui auraient été obtenues par une requête envoyée directement au serveur.

L'architecture première du Web, comme elle est présentée dans le schéma 5-5 [11], a été définie par l'ensemble des contraintes «client/serveur sans état avec cache». En effet, le principe de conception présenté pour l'architecture du Web avant 1994 se concentrait sur des interactions client/serveur sans état pour l'échange de documents statiques via Internet. Les protocoles pour des interactions communicantes avaient certes un support basique de caches non partagés, mais ne contraignaient pas l'interface à un ensemble cohérent de définitions sémantiques pour toutes les ressources. Au lieu de cela, le Web s'est appuyé sur l'utilisation d'une mise en œuvre commune d'une librairie client/serveur (libwww du CERN) afin

de maintenir la consistance à travers les applications du Web.

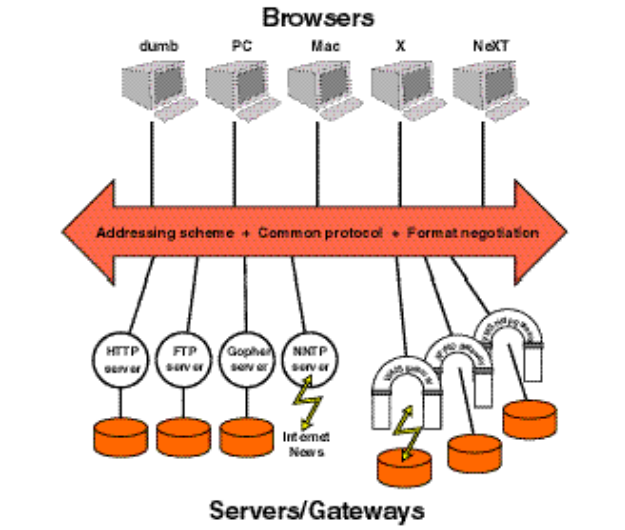


Figure 5-5. Early WWW Architecture Diagram

Schéma 5-5:Architecture première du WWW

Les développeurs d'implémentations Web étaient déjà allés plus loin que cette conception première. En plus des documents statiques, les requêtes pouvaient identifier des services produisant dynamiquement des réponses, telles que les «imagemap» [Kevin Hughes] et les scripts côté serveur [rob McCool]. Des travaux avaient également commencé sur des composants intermédiaires, notamment sur des mandataires *[ndt. proxy]* [79] et sur des caches partagés [59]. Cependant des extensions aux protocoles étaient nécessaires afin de les faire communiquer d'une façon sûre. Les sections suivantes décrivent les contraintes complémentaires ajoutées au modèle architectural du Web afin de guider les extensions conduisant à l'architecture moderne du Web.

5.1.5 Interface uniforme

Le point fondamental qui distingue le modèle d'architecture REST des autres modèles basés sur les concepts réseau est la mise en exergue d'une interface uniforme entre les composants (schéma 5-6). En appliquant le principe logiciel de généralisation à l'interface du composant, l'architecture globale du système est simplifiée et la visibilité des interactions est améliorée. Les mises en œuvre sont découplées des services qu'elles fournissent, ce qui favorise les indépendances des évolutions. Bien sûr, la contrepartie est qu'une interface uniforme pénalise l'efficacité, puisque l'information est transmise dans une forme normalisée plutôt que dans une forme spécifique aux besoins d'une application. L'interface REST est conçue pour être efficace pour le transfert de données hypermédia de granularité importante, optimum pour le cas classique du Web. Mais il en résulte une interface qui n'est pas optimale pour d'autres formes d'interaction architecturale.

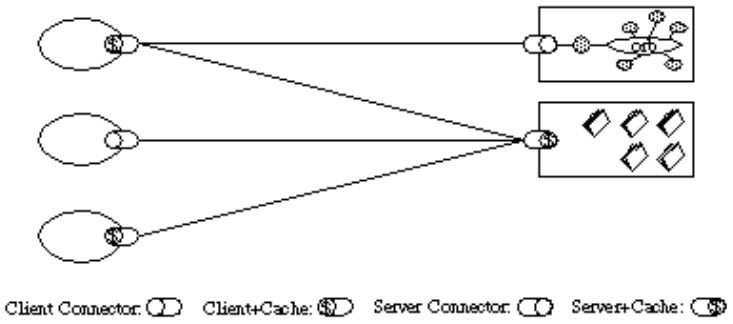


Figure 5-6. Uniform-Client-Cache-Stateless-Server

Schéma 5-6: Le modèle client/serveur avec interface uniforme, cache et sans état

Afin d'obtenir une interface uniforme, de multiples contraintes d'architecture sont nécessaires pour guider le comportement des composants. REST est défini par quatre contraintes d'interface : identification des ressources ; manipulation des ressources par des représentations ; messages auto-descriptifs et l'hypermédia comme moteur de l'état de l'application. Ces contraintes seront abordées dans la section 5.2.

5.1.6 Système en couches

Afin d'améliorer le comportement de l'architecture face aux besoins de grande échelle (internet), nous ajoutons des contraintes de système en couches (le schéma 5-7). Le modèle de système en couches, décrit dans la section 3.4.2, permet à une architecture de se composer de couches hiérarchiques en contraignant le comportement des composants. Chaque composant ne peut pas «voir» au delà de la couche immédiate avec laquelle il interagit. En limitant la connaissance du système à une seule couche, nous mettons une limite sur la complexité du système global et favorisons l'indépendance des couches. Les couches peuvent être employées pour encapsuler des services déjà en place et protéger les nouveaux services des clients existants, en simplifiant les composants par le déplacement de fonctionnalités rarement utilisées dans un espace intermédiaire partagé. Des intermédiaires peuvent également être employés pour améliorer la montée en charge du système en offrant un équilibrage de charges pour les services et ce à travers de multiples réseaux et processeurs.

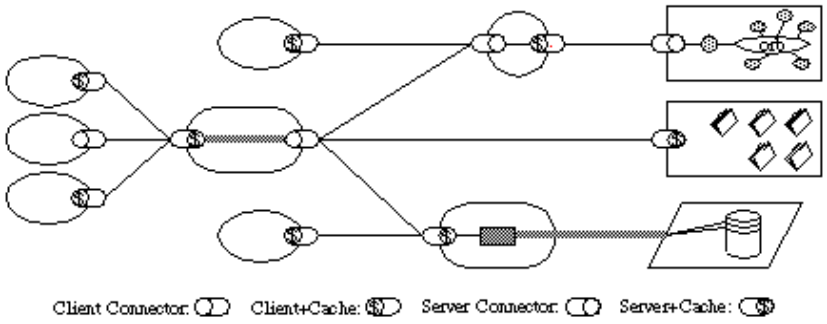


Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

Schéma 5-7: Le modèle client/serveur en couches avec interface uniforme, cache et sans état

Le principal inconvénient des systèmes en couche est qu'ils ajoutent un surcoût et une latence supplémentaire au traitement des données, réduisant ainsi la perception de performance pour l'utilisateur [32]. Pour un système basé sur les concepts réseau qui supporte les contraintes de cache, ce peut être compensé par les bénéfices apportés par du cache partagé aux niveaux intermédiaires. Ajouter des caches partagés aux frontières du domaine d'une organisation peut avoir comme conséquence des gains de performance significatifs [136]. De plus, de telles couches permettent également de renforcer les politiques de sécurité sur les données aux frontières des organisations, comme les pare-feu l'exigent [79].

La combinaison des contraintes amenées par les systèmes en couches et l'unification d'interface conduisent à des propriétés architecturales semblables à celles du modèle «tube + filtre» uniforme (section 3.2.2). Même si les interactions REST sont bidirectionnelles, chaque flux de données grosse maille peut être traité comme un réseau de flux de données, avec des composants de filtrage appliqués de façon sélective à ces flots de données afin de transformer le contenu au fil de l'eau [26]. Dans le contexte REST, les composants intermédiaires peuvent transformer le contenu des messages car ils se décrivent eux-mêmes et leur sémantique est visible par les intermédiaires.

5.1.7 Code à la demande

Le dernier ajout à notre ensemble de contraintes pour REST provient du modèle de code-à-la-demande de la section 3.5.3 (schéma 5-8). REST permet l'extension des fonctionnalités d'un client par le biais de téléchargement et d'exécution de code sous forme d'applet ou de scripts. Cela simplifie les clients en réduisant le nombre de fonctionnalités qu'ils doivent mettre en œuvre par défaut. La possibilité de télécharger des fonctionnalités après le déploiement améliore l'extensibilité du système. Elle réduit cependant la visibilité. De ce fait, elle constitue une contrainte facultative dans REST.

La notion de contrainte facultative peut ressembler à un oxymore. Cependant, elle a un sens dans la conception d'architecture de système qui englobe des frontières d'organisation multiples. Cela signifie que l'architecture ne bénéficie (et souffre des inconvénients) des contraintes optionnelles que lorsqu'elles ont un effet réel pour une certaine partie du système global. Par exemple, s'il est avéré que tous les postes clients d'une organisation supportent les applets Java [45], alors les services de cette organisation peuvent être construits de manière à tirer profit de fonctionnalités étendues apportées par des classes Java téléchargeables. Le pare-feu de l'organisation peut en même temps empêcher le transfert d'applets Java depuis des sources extérieures. Ainsi du point de vue du reste du Web, il sera clair que ces clients ne supportent pas le «code à la demande». Une contrainte optionnelle nous permet de concevoir une architecture qui supporte le comportement désiré dans le cas général, mais avec la possibilité de le désactiver dans certains contextes.

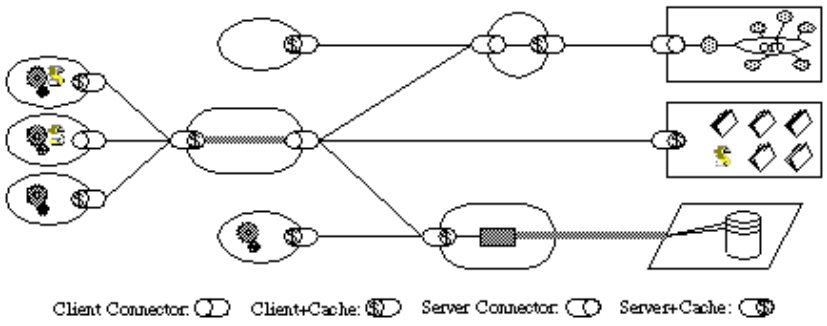


Figure 5-8. REST

Schéma 5-8: Le modèle REST

5.1.8 Résumé de la dérivation de modèle

REST se compose d'un ensemble de contraintes architecturales choisies pour les propriétés qu'elles induisent sur les architectures cibles. Même si chacune de ces contraintes peut être considérée en tant que telle, une description globale de leur dérivation depuis des modèles d'architecture communs facilite la compréhension de la logique derrière leur choix. Le schéma 5-9 montre graphiquement la dérivation des contraintes de REST en se référant aux modèles d'architecture basés sur le réseau, décrits dans le chapitre 3.

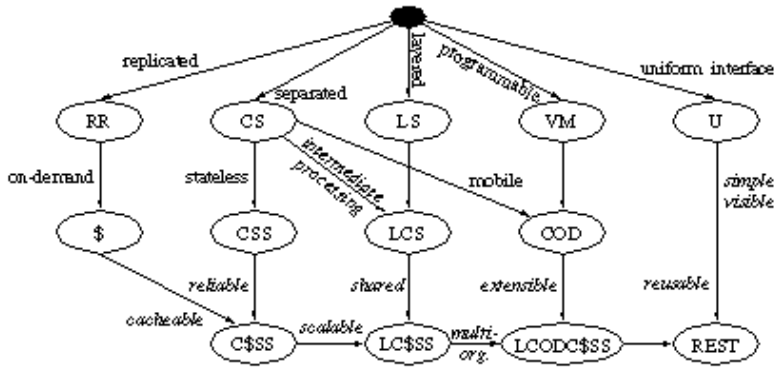


Figure 5-9. REST Derivation by Style Constraints

Schéma 5-9: La dérivation REST par l'application de contraintes

5.2 Eléments architecturaux de REST

Le modèle REST (Representational State Transfer) est une abstraction des éléments architecturaux d'un système réparti d'hypermédias. REST est indépendant des détails de mise en œuvre des composants et de syntaxe de protocole. Il se concentre ainsi sur les rôles des composants, les contraintes sur leur interaction avec d'autres composants, et leur interprétation des éléments de données significatives. Il englobe les contraintes fondamentales sur les composants, les connecteurs et les données qui définissent la base de l'architecture du Web et ainsi l'essence de leur comportement en tant qu'application réseau.

5.2.1 Eléments de données

À la différence du modèle objet distribué [31], dans lequel toutes les données sont encapsulées et cachées par les composants de traitement, la nature et l'état des éléments de données d'une architecture constituent un aspect fondamental de REST. La raison de cette conception provient de la nature des hypermédias distribués. Lorsqu'on choisit un lien, l'information doit être déplacée depuis l'endroit où elle est stockée jusqu'à l'endroit où elle sera utilisée, dans la plupart des cas, par un lecteur humain. Cela marque une différence par rapport aux nombreux autres paradigmes de traitement distribué [6, 50], où il est possible, et habituellement plus efficace, de déplacer «l'agent de traitement» (par exemple, du code mobile, une procédure stockée, une expression de recherche, etc.) vers les données plutôt que de déplacer les données au niveau du processeur.

Face à des hypermédias distribués, un architecte ne possède que trois options de base: 1) effectuer le rendu des données à l'endroit où elles sont localisées et envoyer au destinataire une image dans un format fixe; 2) encapsuler les données avec un moteur de rendu et envoyer les deux au destinataire ; ou, 3) envoyer les données brutes au destinataire avec des métadonnées décrivant leur type, afin qu'il puisse choisir son propre moteur de rendu.

Chaque option possède ses avantages et ses inconvénients. La première, reflétant le modèle traditionnel client/serveur [31], permet de cacher toutes les informations sur la véritable nature des données, empêchant de faire des suppositions quant à la structure des données et facilitant la mise en oeuvre côté client. Elle limite cependant de façon drastique les fonctionnalités du destinataire et implique que la plupart de la charge de traitement soit localisée sur l'expéditeur, conduisant ainsi à des problèmes de montée en charge. La deuxième option, le modèle de l'objet mobile [50], masque les informations tout en permettant des traitements spécialisés sur des données par l'intermédiaire de son moteur de rendu unique. Par contre, elle limite les fonctionnalités du destinataire à ce qui est prévu par le moteur et peut considérablement augmenter la quantité de données transférées. La dernière option permet à l'expéditeur de rester simple et de pouvoir faire face à la charge en réduisant au minimum les octets transférés. Elle perd les avantages du masquage de l'information et demande à l'expéditeur et au destinataire de comprendre les mêmes types de données.

REST propose une vision hybride de ces trois options en se basant sur une compréhension partagée des types de données à l'aide de métadonnées, tout en limitant ce qui est utile à une interface normalisée. Les composants REST communiquent en transférant une représentation d'une ressource dans un format appartenant à un ensemble évolutif de types de données standard. Elle est choisie dynamiquement en fonction des possibilités ou des souhaits du destinataire et de la nature de la ressource. Le fait que la représentation soit dans le même format que la source brute ou qu'elle en soit dérivée reste caché par le mécanisme d'interface. On approche ainsi les avantages du modèle mobile d'objet en envoyant une représentation se composant d'instructions dans le format de données standard sur le moteur de rendu encapsulé (par exemple, Java [45]). REST bénéficie donc de la séparation des concepts du modèle client/serveur sans être confronté au problème de montée en charge du serveur. REST permet de cacher l'information par une interface générique afin de permettre l'encapsulation et l'évolution des services. Ce modèle fournit également un ensemble de diverses fonctionnalités par le biais de moteurs téléchargeables.

Le tableau 5-1 résume les éléments de données de REST.

Tableau 5-1 : Éléments de REST

Éléments d'informations	Exemples modernes de Web
Ressource	la cible conceptuelle d'une référence hypertexte
Identifiant de ressource	URL, URN
Représentation	Document HTML, image JPEG
méta-données de la représentation	type de média, date de dernière modification
méta-données de la ressource	lien sur la source, liens alternatifs, variation
Données de contrôle	if-modified-since, cache-control

5.2.1.1 Ressources et identifiants de ressource

L'abstraction principale de l'information dans REST est la ressource. Toute information pouvant être nommée peut être une ressource : un document ou une image, un service temporel (par exemple «le temps d'aujourd'hui à Marseille»), une collection d'autres ressources, un objet non-virtuel (par exemple une personne), ainsi de suite. En d'autres termes, tout concept pouvant être la cible d'une référence hypertexte d'un auteur doit entrer dans la définition d'une ressource. C'est une correspondance conceptuelle à un ensemble d'entités et ce n'est pas l'entité correspondant à cette association à un moment particulier dans le temps.

Plus précisément, une ressource ***R*** est une fonction $\boldsymbol{M_R(t)}$ relationnelle *[ndt. membership]* variant dans le temps, dans laquelle ***R*** correspond à un instant ***t*** à un ensemble d'entités, ou de valeurs, équivalentes. Les valeurs dans cet ensemble peuvent être des ***représentations de ressource*** et/ou des ***identifiants de ressource***. Une ressource peut correspondre à l'ensemble vide. Ceci permet de lier des références à un concept avant même qu'une réalisation de ce concept n'existe -- cette notion était étrangère à la plupart des systèmes hypertextes avant le Web [61]. Certaines ressources sont statiques dans le sens où, lorsqu'on les examine à n'importe quel instant après leur création, elles correspondent toujours au même ensemble de valeurs. D'autres au contraire ont un degré élevé de variabilité avec le temps. Pour une ressource, le seul critère nécessairement statique concerne la sémantique de correspondance. C'est en effet cette sémantique qui distingue une ressource d'une autre.

Par exemple, la «version plébiscitée par des auteurs» d'un papier d'universitaire est une association dont la valeur change avec le temps, alors que ce qui est associé au «papier publié lors de la conférence X» est statique. Ce sont deux ressources distinctes, même si elles peuvent correspondre toutes les deux à la même valeur à un instant donné. Cette distinction est nécessaire. Ainsi les deux ressources peuvent être identifiées et référencées indépendamment. L'identification distincte d'un fichier source issu d'un gestionnaire de configuration est un exemple similaire dans l'ingénierie logicielle. Cette identification peut se rapporter à la «dernière révision», à la «révision portant le numéro 1.2.7» ou encore à la «révision incluse avec la livraison orange.»

Cette définition abstraite d'une ressource fournit les concepts clés de l'architecture du Web. Tout d'abord, elle permet de généraliser en enveloppant de nombreuses sources d'information sans les distinguer artificiellement par leur type ou leur mise en oeuvre. Ensuite, elle permet de faire une liaison très tardive entre une référence

et une représentation, permettant ainsi une négociation de contenu se basant sur les caractéristiques de la demande. Enfin, elle permet à un auteur de mettre en exergue un concept plutôt qu'une représentation donnée de ce concept. Ainsi, la nécessité de changer tous les liens existants à chaque fois que la représentation change n'existe plus (en partant du principe que l'auteur a utilisé le bon identifiant).

REST utilise un identifiant de ressource pour identifier la ressource particulière impliquée dans une interaction entre les composants. Les connecteurs REST fournissent une interface générique pour accéder et manipuler l'ensemble de valeurs d'une ressource, indépendamment de la façon dont la fonction relationnelle est définie ou du type de logiciel qui manipule la requête. L'autorité de nommage ayant assigné l'identifiant à la ressource, lui permettant d'être référencée, est responsable du maintien de la validité sémantique des correspondances dans le temps (c.-à-d. en s'assurant que la fonction définissant la relation ne change pas).

Les systèmes hypertextes traditionnels [61], fonctionnant typiquement dans un environnement fermé ou local, emploient des identifiants uniques de noeud ou de document qui évoluent au rythme du changement de l'information, en s'appuyant sur des serveurs de liens afin de maintenir les références indépendamment du contenu [135]. Les serveurs centralisés de liens se révèlent être une hérésie face aux exigences du web dictées par des domaines d'information à grande échelle et multi-organisationnels. REST s'appuie plutôt sur l'auteur qui choisit un identifiant de ressource qui caractérise au mieux la nature du concept identifié. Naturellement, la qualité d'un identifiant est souvent proportionnelle à la somme d'argent engagée pour maintenir sa validité, ce qui conduit à des liens cassés lorsqu'une information éphémère (ou mal supportée) est déplacée ou détruite dans le temps.

5.2.1.2 Représentations

Les composants REST effectuent des actions sur une ressource en utilisant une représentation pour capturer l'état courant ou prévu de cette ressource et en transférant cette représentation entre les composants. Une représentation est une séquence d'octets, plus des métadonnées qui les décrivent. Des noms comme document, fichier, entité de message HTTP, instance ou variante sont utilisés pour désigner une représentation de façon générale mais sont moins précis.

Une représentation se compose de données, de métadonnées décrivant les données, et occasionnellement, de métadonnées décrivant les métadonnées (en principe afin de vérifier l'intégrité des messages). Ces métadonnées sont sous forme de paires nom/valeur, où le nom correspond à un standard qui définit la structure et la sémantique de la valeur. Les messages de réponse peuvent inclure des métadonnées de représentation et des métadonnées de ressource : informations sur la ressource qui ne sont pas spécifiques à la représentation fournie.

Les données de contrôle définissent la nature d'un message entre les composants, comme l'action demandée ou la signification d'une réponse. Ils sont également utilisés pour paramétrer les requêtes et surcharger le comportement par défaut de certains éléments communicants. Par exemple, le comportement du cache peut être modifié par des donnés de contrôle incluses dans le message de la requête ou de la réponse.

Selon ces données de contrôle du message, une représentation peut indiquer l'état

courant ou l'état désiré de la ressource demandée, ou la valeur d'autres ressources, comme la représentation des données d'entrée du formulaire ou encore une représentation d'une certaine condition d'erreur pour une réponse. Par exemple, la modification à distance d'une ressource exige que l'auteur envoie une représentation au serveur, établissant de ce fait une valeur pour cette ressource qui peut être recherchée via des requêtes postérieures. Si l'ensemble des valeurs d'une ressource se compose à un instant donné de représentations multiples, la négociation de contenu peut être utilisée pour choisir la meilleure représentation afin de l'inclure dans un message donné.

Le format de données d'une représentation est connu comme étant un type de média [48]. Une représentation peut être incluse dans un message et être traitée par le destinataire en fonction des donnés de contrôle du message et de la nature du type de média. Certains types de média sont destinés au traitement automatisé, d'autres sont prévus pour un rendu pour un utilisateur. Seul un petit nombre sont capables des deux. Des types de média composés peuvent alors être utilisés pour rassembler des représentations multiples dans un message simple.

La conception d'un type de média peut avoir un impact direct sur la perception de performance d'un système réparti d'hypermédia que peut avoir l'utilisateur. Chaque donnée qui doit être reçue avant que le destinataire puisse commencer le processus de rendu de la représentation s'ajoute à la latence de l'interaction. Un format de données qui place l'information de rendu la plus importante au tout début, de telle sorte que l'information initiale puisse être rendue de façon incrémentale tandis que le reste de l'information continue d'arriver, a pour résultat une meilleure perception de performance par l'utilisateur par rapport à un format de données devant être entièrement reçues avant de commencer le rendu.

Par exemple, un navigateur web qui peut mettre en forme progressivement un document HTML volumineux alors qu'il continue de le recevoir fournit une meilleure perception de performance à l'utilisateur par rapport à un autre qui attend que le document entier soit complètement reçu avant de l'afficher, et cela même si les performances du réseau sont les mêmes. Notez que la capacité de rendu d'une représentation peut également être impactée par le choix du contenu. Si les dimensions des tables calculées dynamiquement et des objets inclus doivent être déterminées avant qu'elles puissent être rendues, leur occurrence dans la zone d'affichage d'une page hypermédia augmentera sa latence.

5.2.2 Connecteurs

REST emploie divers types de connecteur, récapitulés dans le tableau 5-2, afin d'encapsuler les activités d'accès aux ressources et de transfert des représentations de ressource. Les connecteurs offrent une interface abstraite pour la communication entre composants, amenant de la simplicité en fournissant une séparation claire des problèmes et en cachant la mise en œuvre sous-jacente des ressources et des mécanismes de communication. La généralisation du concept d'interface permet également la substituabilité : si, pour les utilisateurs, le seul accès au système se fait par l'intermédiaire d'une interface abstraite, alors la mise en œuvre peut être remplacée sans les affecter. Comme un connecteur gère la communication réseau pour un composant, l'information peut être partagée au travers de multiples interactions afin d'améliorer l'efficacité et les temps de réponse.

Tableau 5-2 : Connecteurs REST

Connecteur	Exemples Web
client	libwww, libwww-perl
serveur	libwww, Apache API, NSAPI
cache	cache du navigateur, réseau de cache d'Akamai
résolveur	bind (bibliothèque de consultation DNS)
tunnel	SOCKS, SSL après HTTP CONNECT

Toutes les interactions REST sont sans état. C'est-à-dire que chaque requête contient toutes les informations nécessaires pour qu'un connecteur puisse comprendre la demande et ce indépendamment de toutes les requêtes qui ont pu l'avoir précédées. Cette restriction accomplit quatre fonctions : 1) elle enlève tout besoin pour les connecteurs de maintenir l'état de l'application entre les requêtes, réduisant ainsi la consommation de ressources physiques et améliorant la montée en charge; 2) elle permet à des interactions d'être traitées en parallèle sans exiger de compréhension sémantique par le mécanisme de traitement; 3) elle permet à un intermédiaire de regarder et de comprendre une requête de façon isolée, ce qui peut être nécessaire quand des services sont modifiés dynamiquement; et, 4) elle force toutes les informations qui pourraient être factorisées dans la réutilisation d'une réponse en cache à être présente dans chaque requête.

L'interface des connecteurs est semblable à une invocation procédurale, mais avec des différences importantes dans le passage des paramètres et des résultats. Les paramètres en entrée se composent de données de contrôle de la requête, d'un identifiant de ressource indiquant la cible de la requête et d'une représentation facultative. Les paramètres en sortie se composent de données de contrôle de la réponse, de métadonnées optionnelles sur la ressource et d'une représentation facultative. D'un point de vue abstrait l'invocation est synchrone, mais les paramètres en entrée et en sortie peuvent être passés comme des flux de données. En d'autres termes, le traitement peut être invoqué avant que la valeur des paramètres ne soit complètement connue, évitant ainsi la latence du traitement par lot d'un gros volume de données transférées.

Les premiers types de connecteur sont les connecteurs client et serveur. La différence essentielle entre les deux est qu'un client initie la communication en faisant une demande, tandis qu'un serveur est à l'écoute de connexions et répond aux requêtes afin d'assurer l'accès à ses services. Un composant peut inclure des connecteurs à la fois client et serveur.

Un troisième type de connecteur, le connecteur de cache, peut être situé sur l'interface d'un connecteur client ou serveur afin de conserver des réponses, pouvant être mises en cache, relatives aux interactions actuelles, de sorte qu'elles puissent être réutilisées pour des requêtes ultérieures. Un cache peut être employé par un client pour éviter la répétition de communication réseau, ou par un serveur pour éviter de répéter le processus de production d'une réponse. Dans les deux cas, il sert à réduire la latence des interactions. Un cache est typiquement mis en oeuvre dans l'espace d'adresse du connecteur qui l'utilise.

Certains connecteurs de cache sont partagés. Cela signifie que des réponses en cache peuvent être utilisées par un client autre que celui pour lequel la réponse initiale a été obtenue. Un cache partagé peut être efficace pour réduire l'impact «d'accès massifs instantanés» sur la charge d'un serveur populaire, en particulier lorsque le cache est organisé hiérarchiquement afin de couvrir un grand nombre de

groupes d'utilisateurs, comme c'est le cas avec l'Intranet d'une entreprise, avec des clients d'un fournisseur de services sur internet (Internet Service Provider), ou encore avec des universités partageant un réseau fédérateur national. Cependant, un cache partagé peut également aboutir à des erreurs si la réponse en cache ne correspond pas à celle qui aurait été obtenue par une nouvelle requête. REST essaie d'équilibrer le souhait de transparence dans le comportement d'un cache avec le celui d'un usage efficace du réseau, plutôt que de supposer qu'un usage transparent est systématiquement nécessaire.

Un cache peut déterminer la possibilité de garder une réponse en cache car l'interface est générique et non spécifique à chaque ressource. Par défaut, la réponse à une requête de récupération peut être mise en cache et les réponses aux autres demandes peuvent ne pas l'être. Si une certaine forme d'authentification de l'utilisateur fait partie de la demande, ou si la réponse indique qu'elle ne devrait pas être partagée, alors la réponse peut être mise en cache uniquement dans un cache non-partagé. Un composant peut surcharger ce comportement par défaut en incluant des données de contrôle marquant l'interaction comme pouvant être mise ou non en cache ou encore seulement pendant une durée limitée.

Un résolveur traduit les identifiants partiels ou complets de ressource en information d'adresse réseau nécessaire pour établir une connexion entre composants. Par exemple, la plupart des URI considèrent qu'un nom d'hôte DNS est le mécanisme pour identifier l'autorité de nommage de la ressource. Afin d'initier une demande, un navigateur web extraira le nom de l'hôte à partir de l'URI et se servira d'un résolveur DNS pour obtenir l'adresse du Protocole Internet (IP) pour cette autorité. D'autres schémas d'identification (par exemple, URN [124]) ont besoin d'un intermédiaire pour traduire un identifiant permanent en une adresse plus volatile afin d'accéder à la ressource identifiée. L'utilisation d'un ou plusieurs mécanismes intermédiaires de résolution peut améliorer la longévité des références de ressources grâce à l'adressage indirect même si cela ajoute des temps de latence à la requête.

Le tunnel est la dernière forme de connecteur. Il relaie simplement la communication à travers une connexion ayant des limites, comme un pare-feu ou une passerelle réseau de bas niveau. La seule raison qu'il soit un élément de REST et non abstrait en tant qu'élément de l'infrastructure réseau est que certains composants REST peuvent dynamiquement commuter du comportement de composant actif à celui de tunnel. L'exemple principal est un serveur HTTP mandataire *[ndt. proxy]* qui bascule en un tunnel en réponse à une requête CONNECT [71]. Cela permet ainsi à son client de communiquer directement avec le serveur distant en utilisant un protocole différent, comme TLS, qui lui n'autorise pas de serveur mandataire. Le tunnel disparaît lorsque les deux extrémités terminent leur communication.

5.2.3 Composants

Les composants REST, récapitulés dans le tableau 5-3, sont typés par leur rôle dans une action globale d'application.

Tableau 5-3 : Composants REST

Composant	Exemples modernes de Web
serveur d'origine	httpd d'Apache, Microsoft IIS
passerelle	Squid, CGI, Reverse Proxy

serveur mandataire	proxy CERN, proxy Netscape, Gauntlet
agent utilisateur	Navigateur Netscape, Lynx, MOMspider

Un agent utilisateur utilise un connecteur client pour initier une requête et devient le destinataire final de la réponse. L'exemple le plus commun est un navigateur web, qui permet d'accéder aux services d'information et fournit les réponses du service selon les besoins de l'application.

Un serveur d'origine utilise un connecteur serveur pour régir l'espace de noms pour une ressource demandée. C'est la source définitive de représentations de ses ressources et il doit être le destinataire final de toute requête qui prévoit de modifier la valeur de ses ressources. Chaque serveur d'origine fournit une interface générique à ses services via une hiérarchie de ressources. Les détails de mise en oeuvre de la ressource sont cachés derrière l'interface.

Les composants intermédiaires agissent en tant que client et serveur afin de faire suivre, avec une traduction éventuelle, des requêtes et des réponses. Un composant mandataire est un intermédiaire choisi par un client pour fournir une encapsulation d'interface à des autres services, traduction de données, amélioration de performance ou protection de sécurité. Un composant passerelle (c'est-à-dire un serveur mandataire inverse) est un intermédiaire imposé par le réseau ou le serveur d'origine pour fournir une encapsulation d'interface à d'autres services, pour la traduction de données, l'amélioration des performances ou l'application de la sécurité. Notez que la différence entre un serveur mandataire et une passerelle réside dans le fait qu'un client détermine quand il utilisera un serveur mandataire.

5.3 Vues architecturales de REST

Maintenant que nous avons une compréhension unitaire des éléments d'architecture composant REST, nous pouvons utiliser des vues architecturales [105] pour décrire comment les éléments fonctionnent ensemble pour former une architecture. Trois types de vue -- processus, connecteur et données -- sont utiles pour éclairer les principes de conception REST.

5.3.1 Vue Processus

La vue processus d'une architecture est intéressante principalement pour obtenir les relations d'interaction entre les composants en indiquant le chemin que parcourent les données à travers le système. Malheureusement, l'interaction d'un vrai système implique habituellement un nombre très important de composants, produisant une vue globale croulant sous les détails. Le schéma 5-10 montre un exemple d'une vue processus d'une architecture basée sur REST lors du traitement de trois requêtes parallèles.

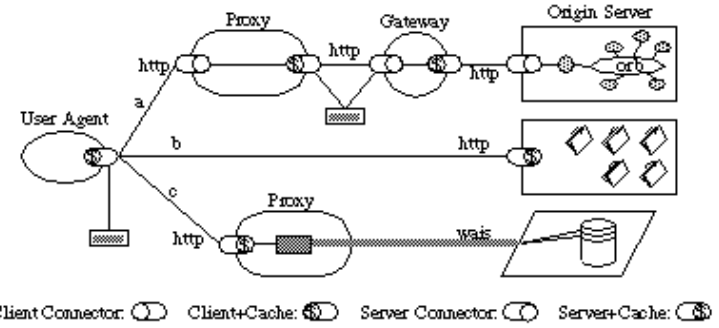


Figure 5-10. Process View of a REST-based Architecture

A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

Schéma 5-10: Vue processus d'une architecture REST

La séparation des concepts client/serveur, amenée par REST, simplifie la mise en oeuvre des composants, réduit la complexité de la sémantique des connecteurs, améliore l'efficacité des réglages d'amélioration des performances et augmente la possibilité de montée en charge des composants purement serveur. Les contraintes qu'amène un système en couches autorisent l'utilisation d'intermédiaires -- serveurs mandataires, passerelles et pare-feu -- pouvant être introduits à divers endroits dans la communication sans pour autant changer les interfaces entre les composants. Ainsi, ils peuvent aider à la traduction dans la communication ou améliorer les performances par l'intermédiaire de vastes caches partagés. REST permet les traitements intermédiaires en contraignant les messages à être auto-descriptifs : l'interaction est sans état entre les requêtes, des méthodes standard et des types de média sont utilisés pour indiquer la sémantique et l'échange d'information, et les réponses indiquent explicitement la possibilité de résider en cache.

Les composants étant connectés dynamiquement, leur arrangement et leur fonctionnement pour une action particulière d'une application possède des caractéristiques semblables à un modèle «tube + filtre» *[ndt. pipe-and-filter]*. Bien que les composants REST communiquent par l'intermédiaire de flux bidirectionnels, le traitement de chaque direction est indépendant et donc modifiable par des transducteurs de flux (filtres). L'interface générique des connecteurs permet à des composants d'être placés sur le flux en se basant sur les propriétés de chaque requête ou réponse.

Les services peuvent être mis en oeuvre en utilisant une hiérarchie complexe d'intermédiaires et de multiples serveurs distribués. La nature sans état de REST permet à chaque interaction d'être indépendante des autres, supprimant le besoin de connaissance de la topologie globale des composants, ce qui constitue une tâche impossible pour une architecture à l'échelle d'Internet. Elle permet aussi à des composants d'agir en tant que destinataires ou intermédiaires, ce choix étant déterminé dynamiquement par la cible de chaque requête. Les connecteurs doivent seulement se connaître mutuellement pendant la durée de leur communication,

même s'ils peuvent cacher l'existence et les possibilités d'autres composants pour des raisons de performance.

5.3.2 Vue connecteur

Une vue connecteur d'une architecture se concentre sur les mécanismes de communication entre les composants. Dans le cas d'une architecture basée sur REST, nous nous intéressons particulièrement aux contraintes qui définissent l'interface générique de ressource.

Les connecteurs client examinent l'identifiant de ressource afin de choisir un mécanisme de communication appropriée pour chaque requête. Par exemple, un client peut être configuré pour se connecter à un serveur mandataire spécifique, peut-être à celui qui agit en tant que filtre d'annotation, lorsque l'identifiant indique que c'est une ressource locale. De même, un client peut être configuré pour rejeter des demandes d'un certain sous-ensemble d'identifiants.

REST ne limite pas la communication à un protocole particulier, mais il contraint l'interface entre les composants, et par conséquent la portée des interactions et les hypothèses de mise en œuvre qui pourraient être faites entre les composants. Par exemple, le protocole Web de transfert principal est HTTP, mais l'architecture inclut également l'accès sans échec aux ressources présentes sur les serveurs réseau préexistants, comme FTP [107], Gopher [7] et WAIS [36]. L'interaction avec ces services est limitée à la sémantique d'un connecteur REST. Cette contrainte sacrifie certains des avantages d'autres architectures, comme l'interaction avec état d'un protocole avec feedback comme WAIS, afin de maintenir les avantages d'une interface simple et générique pour la sémantique des connecteurs. En retour, l'interface générique permet d'accéder à une multitude de services via un mandataire simple. Si une application a besoin de possibilités complémentaires d'une autre architecture, elle peut les mettre en oeuvre et les appeler comme un système séparé fonctionnant en parallèle, de manière similaire à la façon dont l'architecture Web s'interface avec des ressources «telnet» et «mailto».

5.3.3 Vue orientée données

La vue orientée données d'une architecture indique l'état d'une application lorsque l'information traverse les composants. REST étant spécifiquement destiné aux systèmes d'informations répartis, il envisage une application comme une structure cohésive d'informations et d'alternatives dans laquelle un utilisateur peut exécuter une tâche. Par exemple, la recherche d'un mot dans un dictionnaire en ligne est une application, de même qu'un voyage dans un musée virtuel, ou la révision d'un ensemble de notes scolaires en vue de passer un examen. Chaque application définit des buts pour le système sous-jacent, qui serviront à mesurer la performance du système.

Les interactions entre composants se produisent sous forme de messages taillés dynamiquement. Les messages fins ou de granularité moyenne sont employés pour la sémantique des commandes, mais la grosse partie du travail d'une application est accomplie par l'intermédiaire de messages à granularité importante, messages qui contiennent une représentation complète des ressources. La forme la plus fréquente de sémantique de requête est celle permettant d'obtenir une représentation d'une ressource (par exemple, la méthode «GET» dans HTTP), qui peut souvent être mise en cache pour une réutilisation ultérieure.

REST concentre tous les états de contrôle dans les représentations reçues en réponse aux interactions. Le but est d'améliorer la montée en charge du serveur en éliminant tout besoin pour le serveur de maintenir une connaissance de l'état du client au delà de la requête courante. L'état d'une application est donc défini par ses requêtes en suspens, la topologie des composants connectés (certains peuvent filtrer des données bufferisées), les requêtes actives sur ces connecteurs, les flux de données des représentations en réponse à ces demandes et le traitement de ces représentations lorsqu'elles sont reçues par l'agent utilisateur.

Une application atteint un état stable chaque fois qu'elle n'a aucune requête en attente ; c'est-à-dire qu'elle n'a aucune demande en suspens et que toutes les réponses à son ensemble courant de requêtes ont été complètement reçues ou reçues au point où elles peuvent être traitées comme un flux de données de représentation. Pour un navigateur, cet état correspond à une «page Web», qui comprend la représentation principale ainsi que les représentations auxiliaires, comme les images intégrées, les applets embarquées et les feuilles de style. L'importance des états stables des applications a des impacts visibles sur la performance perçue par l'utilisateur et le volume du trafic réseau.

La perception de performance d'un navigateur du point de vue de l'utilisateur est déterminée par la latence entre les états stables : le moment entre la sélection d'un lien hypermédia sur une page Web et celui à partir duquel l'information utilisable a été rendue pour la prochaine page Web. L'optimisation des performances d'un navigateur est donc centrée sur la réduction de cette latence de communication.

Les architectures basées sur REST communiquant principalement par le transfert de représentations de ressources, la conception des protocoles de transmission et la conception des formats de données de représentation peuvent toutes deux avoir un impact sur cette latence. La capacité de rendre de façon incrémentale les données de la réponse alors qu'elle est reçue est déterminée par la conception du type de média et par la disponibilité d'information de rendu (dimensions visuelles des objets intégrés) de chaque représentation.

Une observation intéressante est que la requête réseau la plus efficace est celle qui n'emploie pas le réseau. En d'autres termes, la capacité de réutiliser une réponse depuis un cache a pour conséquence une amélioration considérable des performances de l'application. Même si l'utilisation d'un cache ajoute de la latence à chaque requête individuelle à cause du surcoût de la recherche, la latence moyenne de la demande est sensiblement réduite même si seulement un petit pourcentage des requêtes n'est éligible au cache.

Le prochain état de contrôle d'une application réside dans la représentation de la première ressource demandée. Ainsi l'obtention de cette première représentation est une priorité. L'interaction REST est donc améliorée par les protocoles qui «répondent d'abord et pensent plus tard.» Dit autrement, un protocole qui exige de multiples interactions pour chaque action d'un utilisateur, afin d'entreprendre des négociations sur les possibilités du dispositif avant d'envoyer un contenu de réponse, sera perçu comme étant plus lent qu'un protocole qui envoie ce qui est susceptible d'être le plus optimal dans un premier temps, puis fournit une liste de solutions alternatives dans laquelle le client peut rechercher si la première réponse est insuffisante.

L'état de l'application est contrôlé et stocké par l'agent utilisateur et peut se

composer de représentations provenant de multiples serveurs. En plus de libérer le serveur des problèmes de monter en charge pour le stockage de l'état, cela permet à l'utilisateur de manipuler directement cet état (historique d'un navigateur web, par exemple), de prévoir des changements de cet état (par exemple, des cartes de liens et pré remplissage de représentations) et de sauter d'une application à une autre (signets et dialogues avec des entrées contenant des URI par exemple).

L'application modèle est donc un moteur qui navigue d'un état vers le suivant en examinant et en choisissant parmi les transitions d'état alternatives dans l'ensemble courant des représentations. Il n'est pas surprenant de constater que cela correspond exactement à l'interface utilisateur d'un navigateur d'hypermédia. Cependant, le modèle ne suppose pas que toutes les applications sont des navigateurs. En fait, les détails de l'application sont cachés du serveur par l'interface générique du connecteur. Un agent utilisateur pourrait ainsi être de façon équivalente un robot automatisé effectuant de la récupération documentaire pour un service d'indexation, un agent personnel recherchant des données correspondant à certains critères, ou encore un logiciel de maintenance occupé à parcourir l'information à la recherche de références cassées ou de contenu modifié [39].

5.4 Autres travaux abordant le sujet

Bass, et al. [9] ont consacré un chapitre sur l'architecture du World Wide Web, mais leur description ne couvre que la mise en œuvre de l'architecture développée par le CERN/W3C à travers libwww (bibliothèques client et serveur) et le logiciel Jigsaw. Bien que ces réalisations reflètent plusieurs des contraintes de conception de REST et qu'elles ont été développées par des personnes familières de la conception architecturale et de la logique du Web, la véritable architecture WWW est indépendante de toute mise en oeuvre. Le Web moderne est défini par ses interfaces et des protocoles standard, et non par la façon dont ces interfaces et protocoles sont mis en application dans un morceau donné de logiciel.

Le modèle REST se base sur beaucoup de paradigmes de processus distribués déjà existants [6, 50], sur les protocoles de communication et sur le champ des logiciels. Les interactions de composants REST sont structurées en un modèle client/serveur en couches, mais les contraintes supplémentaires de l'interface générique de ressource créent l'opportunité pour la substituabilité et l'inspection par des intermédiaires. Les requêtes et les réponses ont l'aspect d'un modèle d'invocation à distance, mais les messages REST visent une ressource conceptuelle plutôt qu'un identifiant de mise en oeuvre particulière.

Plusieurs tentatives ont essayé de modéliser l'architecture du Web comme une forme de système de fichiers répartis (par exemple, WebNFS) ou comme un système d'objets répartis [83]. Cependant, elles excluent divers types de ressource du Web ou certaines stratégies de mise en œuvre qui sont «non intéressantes» quand leur présence invalide les hypothèses à la base de tels modèles. REST fonctionne bien parce qu'il ne limite pas la mise en œuvre de ressources à certains modèles prédéfinis, permettant à chaque application d'en choisir une qui correspond au mieux à ses propres besoins. REST permet par ailleurs le remplacement d'une réalisation sans impacter l'utilisateur.

La méthode d'interaction consistant à envoyer des représentations de ressources aux composants les consommant a quelques parallèles avec des modèles d'intégration basés sur les événements *[ndt. EBI pour event-based integration]*. La

différence principale est que les modèles EBI sont basés sur les technologies «push». Le composant contenant les états (équivalent à un serveur d'origine dans REST) envoie un événement à chaque changement d'état, même si aucun composant n'est réellement intéressé par tel événement. Dans le modèle REST, les composants consomment habituellement les représentations en allant les quérir. Bien que ce soit moins efficace du point de vue d'un simple client souhaitant surveiller une simple ressource, la portée du Web rend infaisable un modèle «push».

L'utilisation de principe du modèle REST dans le Web, avec sa notion claire de composants, de connecteurs et de représentations, se rapproche étroitement du modèle architectural C2 [128]. Ce modèle supporte le développement d'applications réparties et dynamiques en se concentrant sur l'utilisation structurée des connecteurs afin d'obtenir l'indépendance de substrat. Les applications C2 se basent sur des notifications asynchrones de changements d'état et sur des messages de requête. Comme avec d'autres schémas basés sur les événements, C2 est basé de façon nominale sur une technologie «push», même si l'architecture C2 pourrait fonctionner dans le modèle REST pull en émettant seulement un avis à la réception d'une demande. Cependant, le modèle C2 ne tient pas compte des contraintes relatives aux intermédiaires de REST, comme l'interface générique de ressource, les interactions garanties sans état et le support intrinsèque du cache.

5.5 Résumé

Ce chapitre a présenté le modèle architectural REST (Representational State Transfer) pour les systèmes répartis d'hypermédias. REST fournit un ensemble de contraintes d'architecture qui, appliquées comme un tout, met en exergue la montée en charge des interactions de composants, la généralisation des interfaces, le déploiement indépendant des composants et l'utilisation de composants intermédiaires afin de réduire la latence d'interaction, d'imposer la sécurité et d'encapsuler les systèmes existants. J'ai décrit les principes d'ingénierie logicielle guidant REST et les contraintes d'interaction choisies pour garantir ces principes, tout en les comparant aux contraintes des autres modèles d'architecture.

Le prochain chapitre présente une évaluation de l'architecture REST au travers de l'expérience et des leçons retenues après avoir appliqué REST à la spécification, à la conception et au déploiement de l'architecture moderne du Web. Ce travail a permis d'écrire les spécifications actuelles des standards d'Internet : le protocole HTTP/1.1 (Hypertext Transfer Protocol) et les URI (Uniform Resource Identifiers). Il a également débouché sur la mise en œuvre de cette architecture par le biais de la bibliothèque libwww-Perl et du serveur HTTP d'Apache.